DEMAND AND CAPACITY MANAGEMENT FOR MEAL DELIVERY SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

Ramón Said Auad Pérez

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology

August 2021

DEMAND AND CAPACITY MANAGEMENT FOR MEAL DELIVERY SYSTEMS

Thesis committee:

Dr. Alan Erera
Co-advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Martin Savelsbergh
Co-advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Pascal Van Hentenryck
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Dr. Marlin Ulmer
Carl-Friedrich Gauss Department
Technische Universität Braunschweig

Date approved: July 30, 2021

ACKNOWLEDGMENTS

First and foremost, my infinite gratitude to my wife Maira Moya, for being my unconditional partner in this journey. Your kind heart and devotion to our family have been an invaluable pillar, especially in the hardest moments. We have been supporting each other for more than 8 years, and I truly wish to keep increasing the count along your side.

To my parents Viviana Pérez and Ramon Auad, thank you for the love you give me every day, and for setting a formidable example to follow. Thank you so much for incessantly believing in me from the very beginning, doing whatever it took to giving me access to a great education, and putting as much time as needed in teaching me about life. This achievement is nothing but the outcome of all your effort and dedication. And thanks for also giving me the chance of having such a wonderful and admirable sister, Javiera. Thanks little sister, for being constantly cheering for me while taking care of our parents in my absence despite these crazy and difficult times we are living. I am so proud of the person you have become and of all your achievements, and cannot wait to see the many ones to come.

I would also like to extend my sincere gratitude to my advisors, Martin Savelsbergh and Alan Erera, for believing in my capabilities by entrusting me very challenging problems, and constantly offering their academic guidance and advice through many hours of discussions and insights. I am as well very grateful to the rest of my thesis committee; Alejandro Toriello and Marlin Ulmer for taking the time to review my work and contributing with their invaluable feedback during the last part of my PhD journey; and Pascal Van Hentenryck, who in addition has been a wonderful collaborator and a great mentor. I would be more than happy to work with any of you in the future.

To my friends, some from Chile and some from many other places: thank you so much for all the shared moments that made me remember over and over that the life of a PhD

student can be more fun than just doing research and coursework. In advance, my deepest apologies for not trying to list everyone who I am grateful with for the many good moments we spent together, please understand that the risk of leaving some names out due to my fragile memory is huge. However, I would like to give a special shout out to my friends: Alfredo Torrico, Matías Siebert, Sebastián Pérez, Felipe Lagos, Adolfo Rocco, and Ian Herszterg, for the countless long afternoon conversations and party nights we spent together laughing out loud, but also reflecting on the latest news as well as on deep scientific and philosophical questions. I do not exaggerate when I say that these moments influenced my current mindset and way of seeing life, and that I will certainly treasure all the joy, laugh and discussions we spent together for the rest of my life. I truly hope that we can keep nurturing the friendship we have developed during these last years.

Lastly, I would like to thank the Agencia Nacional de Investigación y Desarrollo, ANID (former CONICYT), for their invaluable and continuous financial support during my time as a PhD student, through their program Doctorado Becas Chile en el Extranjero 2017, grant award number 72180404.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables ix

List of Figures xi

Summary xiii

Chapter 1: Introduction 1

1.1 Meal delivery systems: boom and challenges 1

1.1.1 Demand management 2

1.1.2 Capacity management 3

1.2 Contributions of the thesis 5

Chapter 2: Using simple integer programs to assess capacity requirements and demand management strategies in meal delivery 7

2.1 Introduction 7

2.1.1 Main contributions 8

2.2 Relevant literature 9

2.3 Single depot setting 11

2.3.1 Problem formulation 11

2.3.2 Creation of a time-expanded network 13
2.3.3 Integer programming formulations 16
2.3.4 Incorporating lateness 18
2.3.5 Radius management 20
2.3.6 *L*-star extension 22
2.4 Two-depot setting 23
2.4.1 Problem formulation 23
2.4.2 Time-expanded network construction 25
2.4.3 Integer program formulations 25
2.4.4 Incorporating lateness 27
2.4.5 Radius management 28
2.5 Computational study 29
2.5.1 Minimum fleet size 31
2.5.2 Analysis of lateness via target delivery time 38
2.5.3 Demand management via service radius adjustments 41
2.6 Conclusion 46

Chapter 3: Courier satisfaction in rapid delivery systems using dynamic operating regions 48

3.1 Introduction 48
3.1.1 Main contributions 49
3.2 Relevant literature 50
3.3 Problem definition 52
3.3.1 Notation and main assumptions 54

3.3.2 Performance metrics . . . 57
3.4 A two-stage matching-based rolling horizon algorithm . . . 59
3.4.1 The courier region reshaping step . . . 60
3.4.2 The order-courier assignment step . . . 65
3.5 Experimental results . . . 67
3.5.1 Instance A . . . 70
3.5.2 Instance B . . . 73
3.5.3 Discussion . . . 76
3.6 Conclusion . . . 78

Chapter 4: Dynamic courier capacity acquisition in rapid delivery systems: a deep Q-learning approach . . . 81

4.1 Introduction . . . 81
4.1.1 Main contributions . . . 82
4.2 Relevant literature . . . 82
4.3 Methodology . . . 85
4.3.1 Problem description . . . 85
4.3.2 Markov decision process formulation . . . 87
4.3.3 Deep Q-learning . . . 89
4.3.4 Baseline policies . . . 93
4.3.5 Performance metrics . . . 95
4.4 Experimental results . . . 96
4.4.1 Experimental settings . . . 96
4.4.2 Learning π^{DQN} . . . 99

4.4.3 Performance 102

4.5 Conclusion 103

Appendices 106

Appendix A: Additional material for Chapter 2 107

Appendix B: Additional material for Chapter 3 115

References 117

LIST OF TABLES

2.1 Characterization of orders of numerical example 16

2.2 Parameter values used for fleet size minimization experiments 31

2.3 Average m^* for some single depot configurations 32

2.4 Average m^* for two depots 35

2.5 Average m^* for two depots (without fleet sharing) 35

2.6 Percentage of instances whose optimal number of crossings is strictly positive 37

2.7 Parameter values used for lateness experiments 38

2.8 Parameter values used for demand management experiments 42

2.9 Times at which the service radius may change 42

2.10 Minimum fleet size m needed to serve 80% and 95% of all orders on average, for a single depot 44

2.11 Minimum fleet size m needed to serve 80% and 95% of orders on average, for two depots 46

3.1 Parameter values used for simulating different number of dynamic regions for Instance A 70

3.2 Sets of values employed in the parameter tuning for Instance A 71

3.3 Parameter values used for simulating different number of dynamic regions for Instance B 74

3.4 Sets of values employed in the parameter tuning for Instance B 74

4.1 Tested and selected values for training parameters. Final values are selected based on best average reward obtained in the testing phase. 98

4.2 Tuned values of opc_{max} and opc_{target} for baseline policies 102

4.3 Per-policy average and standard deviation of performance metrics over the 500 test instances 103

A.1 Characterization of orders of numerical example 114

LIST OF FIGURES

2.1 Example of a time-expanded network for an instance with two orders 16

2.2 Sample distributions 31

2.3 Average m^* for a single depot 33

2.4 Average bundle size for a single depot 33

2.5 Average m^* for two depots 34

2.6 Average bundle size for two depots 36

2.7 Average γ for different instance settings 37

2.8 Average fraction of late orders for a single depot 39

2.9 Average fraction of late orders for two depots 40

2.10 Fraction of served orders with $n = 150$ for a single depot 42

2.11 Fraction of served orders with $n = 150$ for two depots 45

3.1 Visualization of instance setups 68

3.2 Partition of restaurants from Instance A into $p = 4$ base courier regions . . . 70

3.3 Comparison of courier statistics between different configurations for Instance A 72

3.4 Comparison of order statistics between different configurations for Instance A 72

3.5 Distributions of order and courier satisfaction metrics for Instance A considering $p \in \{1, 4\}$ regions 73

3.6 Partition of restaurants from Instance B into $p = 9$ base courier regions . . . 74

3.7 Comparison of courier statistics between different configurations for Instance A . . . 76

3.8 Comparison of order statistics between different configurations for Instance B 77

3.9 Distributions of order and courier satisfaction metrics for $p \in \{1, 9\}$ regions 77

4.1 Standard training process in a RL setting [58] . . . 93

4.2 Evolution of performance metrics during training episodes; 500-episode moving average . . . 100

4.3 Evolution of performance metrics during training episodes; 100-episode moving average . . . 101

4.4 Order coverage in scenarios with and without the capability of adding on-demand courier capacity. Adding couriers in an on-demand basis makes possible to prevent capacity shortage when realized order placement deviates from the average pattern . . . 102

4.5 Policy benchmark over 500 test instances; 20-episode moving average . . . 104

A.1 Example of a time-expanded network with 2 depots and 3 orders. Solid arcs are associated to depot 1, and dashed arcs to depot 2 . . . 114

SUMMARY

With the recent boom of the gig economy, urban delivery systems have experienced substantial demand growth. In such systems, orders are delivered to customers from local distribution points respecting a delivery time promise. An important example is a restaurant meal delivery system, where delivery times are expected to be minutes after an order is placed. The system serves orders by making use of couriers that continuously perform pickups and deliveries. Operating such a rapid delivery system can be very challenging, primarily due to the high service expectations and the considerable uncertainty in both demand and delivery capacity.

In this thesis, we study problems that are based on most recent challenges in meal delivery operations, namely (i) demand management in the form of service area control around restaurants, (ii) capacity management considering the satisfaction of couriers, and (iii) on-demand capacity acquisition based on real-time market signals. To contextualize out work, in Chapter 1 we give an overview of the meal delivery process, and comment on its past success and growth expectations for the future. We also discuss the main operational aspects that make meal delivery a challenging problem in logistics, and then motivate our study on demand and capacity management.

In Chapter 2, we seek to answer several questions in meal delivery operations focused on matching the correct levels of supply with demand. In particular, we consider a demand management mechanism in practice used by meal delivery providers to ensure acceptable customer service when the system is low in delivery capacity, which consists on decreasing demand during an operating day by temporarily reducing the delivery area for one or more restaurants. We show that simple demand restriction strategies allow a significantly smaller fleet to meet service requirements. To simplify analysis, we focus on problem geometries that enable the use of stylized mixed integer programs to optimally deploy a fleet of couri-

ers serving large numbers of orders. Applying the proposed framework to several scenarios with one and two depots, we conduct an extensive experimental study of the effects on system performance of (i) allowing courier sharing between multiple depots, (ii) relaxing the delivery deadlines of placed orders, and (iii) restricting demand through limited adjustment of the service area of restaurants. The results demonstrate the potential effectiveness of different dispatch control and demand management mechanisms, in terms of both the required courier fleet size to serve requests and the coverage level of orders.

Chapter 3 in turn, is devoted to the study of capacity management by considering courier satisfaction. This is a critical aspect in terms of retention/loyalty in a highly competitive environment. Under the premise that couriers prefer to operate in relatively small geographic areas to increase their efficiency, we propose the novel concept of *dynamic courier regions*: small operating regions for couriers which can also be dynamically and temporarily expanded to allow delivery capacity to be shared between regions when necessary to keep customer service performance metrics high. We propose an optimization-based rolling horizon algorithm for courier management which handles both region resizing and request assignment decisions. Experimental results for realistic settings show that the proposed algorithm successfully balances customer and courier satisfaction, simultaneously achieving service quality levels that are comparable to those of a single operating region and courier satisfaction metrics that are comparable to those achieved by fixed, inflexible regions.

Lastly, in Chapter 4 we study the problem of dynamically adding extra courier capacity in a rapid delivery system. Delivery providers typically plan courier shifts for an operating period based on demand forecast. However, because of the high demand volatility it may at times during the operating period be necessary to adjust and dynamically add couriers. To address this problem, we propose a deep reinforcement learning approach to obtain a policy that balances the cost of adding couriers and the cost of service quality degradation by an insufficient number of couriers. Specifically, we seek to ensure that a high fraction of orders

is delivered on time and with a small number of courier hours. A computational study shows that when performing corrective capacity adjustments, a learned policy using the proposed framework outperforms policies representing current practice in the meal delivery space, demonstrating the potential of deep learning for solving operational problems in highly stochastic logistic settings.

CHAPTER 1

INTRODUCTION

1.1 Meal delivery systems: boom and challenges

In the last decade, several industries have been disrupted by internet business models, and the food industry is no exception [1]. Already in 2015, the US food delivery market was valued in $11 billions, with a predicted annual compound rate of 16% towards 2023 [42]. Prior to the COVID-19 outburst, restaurant delivery was already projected to grow more than three times faster than on-premise sales [25, 55], and it was anticipated that by 2030 online food delivery would surpass homemade meals, with delivery sales possibly reaching $365 billion worldwide [21, 67]. Nonetheless, as a consequence of the global pandemic, the observed demand for online meal delivery has seemingly surpassed all previous growth estimations, with skyrocketing adoption and sales reaching unprecedented levels [51, 72].

Many factors are responsible for the success of online food delivery (also known as *online meal delivery*), including diners wanting to spend less time cooking [14, 55, 67]. This fact and the development of the internet have deeply transformed the restaurant industry through the rise of online restaurant aggregators - third-party providers of a digital marketplace where diners have the possibility to place orders from a wide variety of restaurants, some of them also being responsible for the delivery of such orders. As a result, 87% of US diners who use these platforms consider that this service has made their life easier, and 31% admit using it at least twice a week [39]. At the same time, 60% of restaurants report an increase in their sales from having the option of delivery, and the ones subscribed to these platforms have reportedly experienced a raise in their sales by 10% to 20% [25, 43]. What

is more, some restaurants are heavy investing on adapting themselves to digital ordering by downsizing their own seating space, including some of the biggest fast-food chains [55, 76].

Operating rapid delivery networks is a challenging problem in modern logistics due to dynamism and uncertainty [8, 47, 73]. On the demand side, customers dynamically place orders to restaurants in the network, with the system often experiencing abrupt variations in order placement rates throughout the day. Moreover, diners expect their food to be delivered usually within the hour while preserving its freshness. Thus, orders must be dispatched within minutes after they are ready, reducing opportunities to consolidate deliveries to multiple diners to reduce cost. Meal delivery network operators need to dynamically match delivery capacity (provided by *drivers* or *couriers*) to demand, and the usual tradeoffs apply. Lack of delivery capacity at some times leads degradation in customer service, while too much capacity at other times can be overly costly due to an excess of underutilized couriers.

This dissertation contributes by providing studies of different strategies to match the effective delivery capacity of dynamic rapid delivery systems, such as meal delivery, to the ongoing demand, including mechanisms that manage demand to match the existing capacity, and strategies that manage existing and new delivery capacity to maintain the service quality of the system.

1.1.1 Demand management

The first chapter of this thesis studies demand management in meal delivery systems. The concept of demand management is better known as revenue management, and has been thoroughly studied in past decades [35, 57]. Its goal is to balance service requirements with the capabilities of a given provider, to ultimately meet customer demand effectively and efficiently; this allows a system to be more proactive to anticipated demand and more

reactive to unexpected demand, ultimately granting a positive effect on profit due to increased sales and customer loyalty [22].

The most well-known applications of demand management include airline, car rental and hotel businesses [2]. Using the airline industry as a reference, the use of demand management in this application presents obvious similarities to the context of delivery applications, e.g., airlines evaluate whether accepting a seat booking at a given fare or rejecting it with the hope of selling it at a higher fare, whereas a delivery planner analyzes whether using the existing delivery capacity to accept or reject a delivery request when it arrives, and if so, selecting its delivery time slot. However, these two settings also possess fundamental differences: while the cost of booking a given seat does not depend on the individual that purchases the air ticket, the cost of accepting a delivery request on a particular time slot relies on the delivery address and time slot of the request and of other accepted orders [16].

Demand management mechanisms are executed by means of pricing and supply allocation. In the case of delivery applications, dynamic pricing is the most studied demand management mechanism [56, 63, 75], while little research has been done in terms of resource allocation.

1.1.2 Capacity management

Chapters 3 and 4 of this thesis focus on studying delivery capacity management in rapid delivery systems. In this context, the system's delivery capacity is given by the available couriers. These are the means employed by the system to deliver the incoming orders, and so they constitute a fundamental aspect of the network operations.

Matching couriers to deliveries in rapid delivery systems can be especially challenging since couriers often participate in these networks with significant autonomy. In many systems, couriers do not need to accept all work assigned to them. It is typically difficult to issue repositioning decisions to couriers who instead move where they believe they are

most likely to find profitable future work orders. And in the long term, several logistics networks are competing for the same couriers and there are few barriers to keep couriers from moving between companies (or working for multiple networks simultaneously). Thus, maintaining high levels of courier satisfaction can be just as important as customer satisfaction for these networks. Hence, we propose a framework that explicitly model courier satisfaction in delivery operations and balances customer and courier satisfactions using re-sizeable operating regions.

Another recurrent challenge in meal delivery operations is the planning of delivery capacity. Determining the courier schedules for a particular day of operations is very difficult as demand is hard to predict both in terms of volume and order placement timing. In practice, network operators schedule a base fleet of couriers prior to the start of an operating day, and later during the operation they typically perform corrective on-demand courier capacity adjustments based on observed system conditions. Although this procedure allows operators to maintain the quality of the service, it can also increase the system's operating costs as on-demand couriers are usually paid a premium for requesting their services at the last minute. Hence, it becomes of great interest to find a mechanism able to determine exactly how many additional resources are needed to achieve a desired service level without incurring in extra costs due to underutilized courier. In turn, this issue raise multiple research questions yet to be answered in the literature, including (i) what features in the system should be considered when deciding whether to enlarge the existing capacity? (ii) how these features affect the decision of acquiring extra capacity? (iii) how to design an on-demand capacity acquisition policy that dictates how much capacity to acquire given an observed system state? Seeking to answer these questions, we propose a machine learning framework to learns a cost-effective on-demand capacity acquisition policy considering a set of key system features.

1.2 Contributions of the thesis

In Chapter 2, we study several questions in meal delivery operations focused on matching the correct levels of supply with demand. In particular, we consider a demand management technique in practice used by meal delivery providers to ensure excellent customer service, which consists on decreasing demand during an operating day by temporarily reducing the delivery area for one or more restaurants. We show that such simple demand restriction strategies allow a significantly smaller fleet to meet service requirements. To simplify analysis, we focus on problem geometries that enable the use of stylized mixed integer programs to optimally deploy a fleet of couriers serving large numbers of orders. Applying the proposed framework to several scenarios with one and two depots, we conduct an extensive experimental study of the effects on system performance of (i) allowing courier sharing between multiple depots, (ii) relaxing the delivery deadlines of placed orders, and (iii) restricting demand through limited adjustment of the coverage of restaurants. The results demonstrate the potential effectiveness of different dispatch control and demand management mechanisms, in terms of both the required courier fleet size to serve requests and the coverage level of orders.

In Chapter 3, we seek to fill a gap in the literature by considering, for the first time, the satisfaction of couriers in rapid delivery, which is critical in terms of retention/loyalty in a highly competitive environment. Under the premise that couriers prefer to operate in relatively small geographic areas to increase their efficiency, we propose the novel concept of *dynamic courier regions*: small operating regions for couriers which can also be dynamically and temporarily expanded to allow delivery capacity to be shared between regions when necessary to keep customer service performance metrics high. We propose an optimization-based rolling horizon algorithm for courier management which handles both region resizing and request assignment decisions. Experimental results for realistic settings demonstrate that the proposed algorithm successfully balances customer and courier satis-

faction, simultaneously achieving delivery times that are comparable to those of a single operating region and courier satisfaction metrics that are comparable to those achieved by fixed, inflexible regions.

In Chapter 4, we study the problem of dynamically adding extra courier capacity in a rapid delivery system, such as meal delivery. Delivery providers typically plan courier shifts for an operating period based on demand forecast. However, because of the high demand volatility it may at times during the operating period be necessary to adjust and dynamically add couriers. We propose a deep reinforcement learning approach to obtain a policy that balances the cost of adding couriers and the cost of service quality degradation by an insufficient number of couriers. Specifically, we seek to ensure that a high fraction of orders is delivered on time and with a small number of courier hours. A computational study shows that a learned policy outperforms policies representing current practice in the meal delivery space, demonstrating the potential of deep learning for solving operational problems in highly stochastic logistic settings.

CHAPTER 2

USING SIMPLE INTEGER PROGRAMS TO ASSESS CAPACITY REQUIREMENTS AND DEMAND MANAGEMENT STRATEGIES IN MEAL DELIVERY

2.1 Introduction

This chapter seeks to better understand the potential of demand management by adjusting the service coverage area (what we refer to later on as radius management). The meal delivery environment we consider is as follows: during a given operating period, diners place orders to restaurants (in the remainder sometimes called depots to stay close to the terminology commonly used in the vehicle routing literature), and the aggregator must assign these orders to couriers in such a way to deliver (most) orders on time (i.e., at or before the delivery time promised to the diner when the order is placed) while minimizing operating costs. The goal of this chapter is to analyze the fundamental relationship between service and cost metrics in these systems. We consider two different models of customer service, one which requires orders to be delivered by a hard deadline and the other which has a target delivery time but allows some fraction of orders to be delivered between the target time and a (later) hard deadline. We measure cost primarily by the number of couriers required during the operating period.

We develop optimization models that seek to determine the minimum number of couriers required to meet service requirements. To simplify analysis, we study the relationship between service and cost in three stylized settings: (i) a single depot at one extreme of a single line segment serving orders that must be delivered at points on the line segment, (ii)

a single depot at the end of multiple line segments serving orders that must be delivered at points on the line segments, and (iii) two depots at the opposite extremes of a line segment serving orders that must be delivered at points on the line segment from a specific depot. For simplicity, we assume that couriers follow the instructions of the decision maker and never reject offered delivery orders. For each of the settings, we provide an integer programming (IP) formulation that assumes perfect information, i.e., order placement times and delivery locations are known in advance. The results from these models allow us to provide insights to the following fundamental questions:

- For a given service requirement, a given number of couriers, and a given order arrival rate, what fraction of orders can be served (i.e., delivered at or before the delivery time promise)?
- For a given service requirement and a given order arrival rate, what is the minimum number of couriers needed to serve all orders?
- For a given service guarantee, a given number of couriers, and a given order arrival rate, what is the largest coverage area that a depot can serve?

2.1.1 Main contributions

To summarize, the main contributions of our research are:

- We develop an IP framework for studying supply and demand management mechanisms for online meal delivery environments.
- We perform an extensive experimental analysis of the fundamental trade-offs in meal delivery operations, which provides valuable insight into the benefits of supply and demand management mechanisms.

The remainder of the chapter is organized as follows. Section 2.2 provides literature rele-

vant to the problem studied. Sections 2.3 and 2.4 respectively present a detailed description of the settings with one and two depots considered in our research, and introduce the associated IP formulations. Section 2.5 summarizes the results of our computational experiments. Finally, Section 2.6 gives concluding remarks.

2.2 Relevant literature

In its most general setting, the problem we study in this work is one in the family of dynamic vehicle routing problems [45] and more specifically is a dynamic pickup and delivery problem (dPDP) [10]. The existing literature on these type of problems is vast and has grown significantly over the past few decades, mainly due to the advances in technology and telecommunication.

One of the applications for recent research in dPDP problems is transport of persons. Examples of such are dial-a-ride, dial-a-flight and ride-sharing. The latter problem is similar to our problem, with a common fleet of drivers that must satisfy transportation requests on short-notice, each characterized by an origin-destination pair with time-based service requirements [3]. Meal delivery problems are also part of the growing research area of dynamic delivery problems (dDP) [47], including same-day delivery problems. The growth of online retail in the last decades has attracted researchers to same-day delivery operations, with focus on both simplified analytic settings [5, 33, 48, 62] and real world situations [34, 47, 61, 65, 73]. A defining characteristic of the dDP class of problems is that once a vehicle is dispatched to satisfy a set of deliveries, adjusting the route does not produce any benefit if travel times and costs do not change.

In the dDP literature, problems can be classified based on the availability of information. *Static* problems are those where all the information about orders and travel times is deterministic and known in advance [5, 48, 73]. *Dynamic* problems on the other hand, consist on settings where orders are revealed over time, and decisions are made only based on

the revealed information [34, 47, 65]. If in addition, some of the parameters follow a probabilistic distribution and such information is available to the decision maker, then the problem is said to be stochastic. In this chapter we study a static meal delivery routing problem

Routing problems with time constraints are reviewed in [41]. Static dDP problems are closely related to the multi-trip VRP with release dates and deadlines (VRP-rdd). In these problems, couriers may perform multiple trips from the depot to serve orders that have an earliest ready time (release date) at the depot. Deliveries to customers must occur before a deadline. Initial work on these problems considers only release dates (VRP-rd). [18] address a problem with capacitated couriers that must serve all orders minimizing total travel time, and propose a hybrid genetic algorithm to find solutions. [52] study the VRP-rdd and develop a path relinking algorithm to minimize the convex sum of the total distance and total positive deviations (delays) from the target order delivery times.

[5] study the VRP-rd with a global deadline and focus on the computational complexity of variants. For problems with a single courier, they provide polynomial solution algorithms for minimizing either the total time to complete all orders or the distance travelled to complete all orders by the global deadline for special-case geometries including when all customers are located on a line with the depot. [48] extend this work to study the single vehicle VRP-rdd with individual order deadlines on simplified geometries and provide polynomial algorithms for solving the problems of minimizing the returning time of the courier after serving every order and minimizing the total distance traveled.

Similar to [5, 48], the problems we analyze also use simplified geometries, but we allow multiple couriers to deliver orders and attempt to optimize different objectives. Furthermore, our framework can model heterogeneous individual deadlines for each order and can also incorporate more complex features. To handle these additional complexities, we use integer programming formulations built on underlying time-expanded networks, directed

networks whose vertices are pairs with both a location and a time point component. The use of these networks allows more flexibility when modeling time dependencies. Some applications of time-expanded networks include service network design [13, 24] and the time dependent TSP with time windows [69]. However, flexibility comes at the cost of efficiency in solving [54].

Research on demand management in rapid delivery based on service area sizing is scarce. In an analytical static setting, [74] studies the effect of the sizing of the service area of a single restaurant on other fundamental metrics related to meal delivery, such as profit, delivery time, service level and courier revenue. A subsequent work by [60] studies the extension of this problem in a dynamic setting where orders are not known until they are placed, and where the service area around the restaurant can be dynamically resized as a response to deviations from expected demand. Using learning techniques, the authors are able to find a service area sizing policy that improves the number of served orders by over 20% compared to employing a fixed service area.

2.3 Single depot setting

2.3.1 Problem formulation

We consider a single depot located at one end, $\tau_0 = 0$, of the line segment $[0, U]$ with $U > 0$. A set of orders, $N = \{1, \ldots, n\}$ is placed on the depot, where each order $j \in N$ specifies:

- A ready time $r_j \in [0, U] \cap \mathbb{Z}$, which defines the earliest time it can be dispatched for delivery;
- A location $\tau_j \in (0, U] \cap \mathbb{Z}$, representing its delivery location measured in travel time from the depot; and
- A due time $Q_j \geq r_j + \tau_j, Q_j \in \mathbb{Z}$ where if order j is not delivered by time Q_j, it is

considered late (and is potentially lost).

Let $\mathbf{T} \equiv [0, T]$ be the operating period. Without loss of generality, we assume $r_1 = 0$ and $r_j \leq r_{j+1}, \forall j \in N$ (with $r_{n+1} \equiv T$). Furthermore, at time r_j an available courier at the depot can be dispatched to deliver j along with any other orders i with $r_i \leq r_j$ (no courier capacity). We assume that the times required for a courier to pick up or deliver orders are negligible when compared to travel times. Thus, given an order set $J \subseteq N$ with $\tau_J \equiv \max_{j \in J}\{\tau_j\}$, a courier can deliver J and return to the depot in time $2\tau_J$. When J includes more than a single order, we say that the orders in J are *bundled*.

Suppose that there are $m \geq 1$ couriers that can make deliveries, each located at the depot at time 0 and required to return after their final delivery by time T. Let $S > 0$ be the (common) maximum acceptable service time for each order, which implies that $Q_j \equiv r_j + S$ for $j \in N$.

In this setting, we consider two optimization problems: (1) maximize the number of orders that can be served on-time given m couriers, and (2) minimize the number of couriers m needed to serve all orders. Formally:

Problem 1 (Order maximization)**.** *Given m identical couriers, find a feasible delivery schedule for each of them that maximizes the total number of orders served, where a feasible delivery schedule for a courier specifies a number of delivery trips, each with a given departure time and a set of orders to deliver, such that all served orders $j \in N$ are ready at the time of departure and are delivered by their due time Q_j.*

Problem 2 (Courier minimization)**.** *Find the minimum number of couriers (and a feasible delivery schedule for each of them) required to serve all orders $j \in N$ by their due time Q_j.*

In the rest of this section we develop a mathematical framework for analyzing these problems which relies on integer programs defined on time-expanded networks.

2.3.2 Creation of a time-expanded network

Before giving a mathematical model for Problems 1 and 2, we provide a useful proposition; proofs for this and later results can all be found in Appendix A.1.

Proposition 1. *Consider an optimal schedule and let $J \subseteq N$ be a set of orders in that schedule with the same dispatch time t. Then, there exists an optimal schedule in which the orders in J are served by a single courier.*

The next result shows how to determine a sufficient finite subset of time points in $\mathbf{T}$ with the property that there exists an optimal schedule that only dispatches couriers at a subset of these points. Consider then the following definition:

Definition 1 (Active order)**.** We say that order j is active at time $t \in \mathbf{T}$ if $t \in \{r_j, r_j + 1, \ldots, Q_j - \tau_j\}$ and it has not yet been dispatched by t. We denote the set of active orders at time t by $A(t)$.

Active orders at time t can be dispatched feasibly. We now introduce a lemma useful when modeling Problems 1 and 2.

Lemma 2. *Given $j \in N$, let $t \in [r_j, r_{j+1})$ be the earliest time that a courier is available for dispatch at the depot. Then there exists an optimal schedule for Problems 1 and 2 in which no courier is dispatched at any time $(t, r_{j+1}) \cap \mathbb{Z}$.*

From Lemma 2 it follows that the only necessary dispatch times at the depot are the ready times $\{r_j\}_{j \in N}$ and the courier return times $r_j + 2\sum_{k \in K} \tau_k$, for some $K \subseteq N$. We denote the set of such time points by $\mathcal{T}_0$.

The time-expanded networks we build also model couriers moving from one order delivery

location to another or back to the depot. To determine which time points are required to model these movement decisions, let $t \in \mathbf{T}$ be a time point such that an optimal solution dispatches a courier from the depot at t with orders $J \subseteq A(t)$, and let $\{\tau_{(i)}\}_{i=1}^{|J|}$ be the locations of orders J sorted in non-decreasing order from the depot such that $\tau_{(1)}$ is closest. Then there exists an optimal solution where the courier visits locations $\tau_{(i)}$ sequentially at times $t + \tau_{(i)}$ for $i = 1, 2, \ldots, |J|$. After visiting location $\tau_{(|J|)}$ the courier returns to the depot, arriving at time $t + 2\tau_{(|J|)}$ either to be dispatched again immediately or, by Lemma 2, to wait until the next order arrival time.

At any dispatch time $t \in \mathcal{T}_0$ at the depot, an optimal solution will either decide not to dispatch a courier or to dispatch a courier with a subset $J' \subseteq A(t)$. The only optimal subsets are those that include *all* orders with locations $\tau_j \leq \tau_{(i^*)}$ where $\tau_{(i^*)}$ is the furthest order in J', and so each order j in the subset is delivered exactly at time $t + \tau_j$.

Thus, it should be clear that these problems can be solved by considering models that include a discrete set of time points, specifically a subset of the time points specified in Proposition 3 below:

Proposition 3. *To solve Problems 1 and 2, it suffices to consider courier schedule decisions at ready times $r_j, j \in N$, at potential return times to the depot $r_j + 2\sum_{k \in K} \tau_k, j \in N, K \subseteq N$, and at potential delivery times at the customers $r_j + \sum_{k \in K} 2\tau_k + \tau_i, j \in N, K \subseteq N, i \in A(r_j + \sum_{k \in K} 2\tau_k)$.*

Each time point of interest is also associated with a specific spatial location: all dispatches occur at the depot $\tau_0 = 0$, while deliveries are performed at locations $x = \tau_j, j \in N$. Consequently, we will define the nodes of our time-expanded network in the form (t, s), representing a location s in the line segment $[0, U]$ and an associated time point t. Algorithm 1 specifies how to produce the complete time-expanded network for these optimization problems. Before continuing the formulation process, we present the following definition.

Algorithm 1 (CREATE_NETWORK)

Input: $N, (r_j, \tau_j, Q_j)_{j \in N}, T$
Output: Directed network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$

$\mathcal{V} \leftarrow \{(r_j, 0)\}_{j \in N}$
$\mathcal{A} \leftarrow \emptyset$
$\mathcal{T}_0 \leftarrow \{r_j\}_{j \in N}$
for $t \in \mathcal{T}_0$ **do**
    Find lowest index $j^* \in N \cup \{n+1\}$ s.t. $t < r_{j^*}$ ▷ $r_{n+1} \equiv T$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t, 0), (r_{j^*}, 0))\}$
    Compute set of active orders $A(t) = \{j \in N : t \geq r_j, t + \tau_j \leq Q_j\}$
    Sort $\{\tau_j\}_{j \in A(t)}$ in non-decreasing order, into $\{\tau_{(i)}\}_{i=1}^{|A(t)|}$
    for $i = 1, \ldots, |A(t)|$ **do**
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + \tau_{(i)}, \tau_{(i)}), (t + 2\tau_{(i)}, 0)\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tau_{(i-1)}, \tau_{(i-1)}), (t + \tau_{(i)}, \tau_{(i)})), ((t + \tau_{(i)}, \tau_{(i)}), (t + 2\tau_{(i)}, 0))\}$ ▷ $\tau_{(0)} \equiv 0$
        $\mathcal{T}_0 \leftarrow \mathcal{T}_0 \cup \{t + 2\tau_{(i)}\}$
return $\mathcal{N} = (\mathcal{V}, \mathcal{A})$

Definition 2 (Depot and non-depot node)**.** A node of a time-expanded network $(t, s) \in \mathcal{V}$ is called *depot node* if its spatial component s corresponds to a depot location; otherwise it is labeled as *non-depot node*.

For an example of the output of Algorithm 1, consider an instance with $T = 6$, $S = 3$, and whose set of orders to be served $N = \{1, 2\}$ is characterized by Table 2.1. The resulting partial time-expanded network is illustrated in Figure 2.1. In the illustration, nodes that are filled and in the horizontal axis correspond to depot nodes; otherwise they are non-depot nodes, representing when and where orders can be delivered. Arcs inbound to a non-depot node that emerge from a depot node correspond to a dispatch; and arcs inbound to a depot node from a non-depot node represent a return. Arcs between depot nodes represent the action of a courier waiting at the depot; arcs between non-depot nodes represent the action of traveling between order destinations. Note that a dispatch from the depot at $t = 3$ to order 2 is not needed since the return node arrived only from already delivering order 2. In this example, a single courier is able to serve both orders when dispatched at time $t = 1$.

Table 2.1: Characterization of orders of numerical example

j	r_j	τ_j	Q_j
1	0	2	3
2	1	1	4

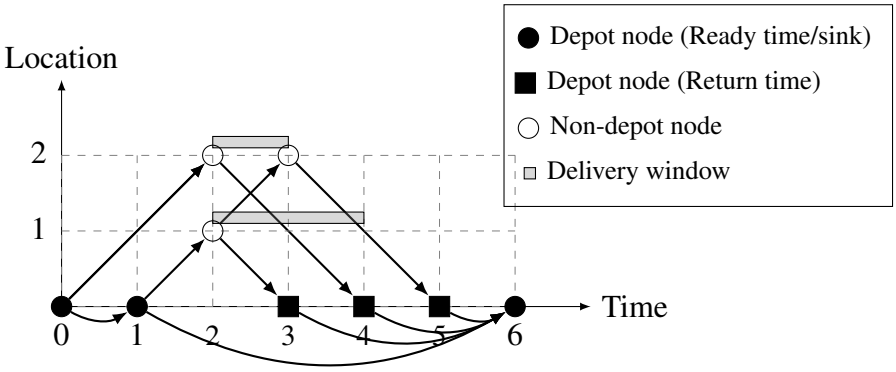


Figure 2.1: Example of a time-expanded network for an instance with two orders

2.3.3 Integer programming formulations

Once the time-expanded network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ is constructed, we can formulate Problems 1 and 2 as integer programs. For each $j \in N$, let $\mathcal{V}_j \equiv \{(t, \tau_j) \in \mathcal{V} : t \in \{r_j + \tau_j, \ldots, Q_j\}\}$ be the set of non-depot nodes at which order j may be delivered, and $\mathcal{V}_0 \equiv \{(t, 0) \in \mathcal{V} : t \in \mathcal{T}_0\}$ be the set of depot nodes (and note that $\mathcal{V} \equiv \bigcup_{j \in N \cup \{0\}} \mathcal{V}_j$). Moreover, for each $p \in \mathcal{V}$, let $\delta_p^- \equiv \{q \in \mathcal{V} : (q, p) \in \mathcal{A}\}$ and $\delta_p^+ \equiv \{q \in \mathcal{V} : (p, q) \in \mathcal{A}\}$.

The decision variables in these problems are:

$$z_{pq} = \text{Number of couriers that traverse arc } (p, q) \in \mathcal{A}$$

$$v_{jp} = \begin{cases} 1 & \text{if order } j \in N \text{ is delivered at node } p \in \mathcal{V}_j \\ 0 & \text{otherwise} \end{cases}$$

For a fixed courier fleet size m, a valid mixed-integer programming formulation for Prob-

lem 1 is given by

$$\max \sum_{j \in N} \sum_{p \in \mathcal{V}_j} v_{jp} \tag{2.1a}$$

$$\text{s.t.} \sum_{p \in \mathcal{V}_j} v_{jp} \leq 1, \ \ \forall j \in N \tag{2.1b}$$

$$v_{jp} \leq \sum_{q \in \delta_p^-} z_{qp}, \ \ \forall j \in N, \ \ \forall p \in \mathcal{V}_j \tag{2.1c}$$

$$\sum_{q \in \delta_{(0,0)}^+} z_{(0,0),q} = m \tag{2.1d}$$

$$\sum_{p \in \delta_{(0,T)}^-} z_{p,(0,T)} = m \tag{2.1e}$$

$$\sum_{p \in \delta_q^-} z_{pq} = \sum_{r \in \delta_q^+} z_{qr}, \ \ \forall q \in \mathcal{V} \setminus \{(0,0),(0,T)\} \tag{2.1f}$$

$$v_{jp} \in \{0,1\}, \forall j \in N, \ \ \forall p \in \mathcal{V}_j \tag{2.1g}$$

$$z_{pq} \in \begin{cases} \mathbb{R}_+ & \text{if } p,q \in \mathcal{V}_0, \\ \{0,1\} & \text{otherwise} \end{cases} \ \forall (p,q) \in \mathcal{A} \tag{2.1h}$$

Objective (2.1a) seeks to maximize the number of served orders. Constraints (2.1b) and (2.1c) are related to order acceptance; each order $j \in N$ can only be delivered once and if this occurs at node $p \in \mathcal{V}_j$, then some courier must travel from a node $q \in \delta_p^-$ to p. Constraints (2.1d) - (2.1f) are courier flow conservation constraints for all the network nodes. Constraints (2.1g) and (2.1h) enforce non-negative flows on depot arcs and binary flows elsewhere (due to Proposition 1).

Using a similar set of constraints and redefining the courier fleet size m as a decision

variable, Problem 2 can be posed as

$$\min \quad m \tag{2.2a}$$

$$\text{s.t.} \sum_{p \in \mathcal{V}_j} v_{jp} = 1, \quad \forall j \in N \tag{2.2b}$$

$$(2.1c) - (2.1h)$$

$$m \in \mathbb{R}_+ \tag{2.2c}$$

Note that Model (2.1) and (2.2) are always feasible. Moreover, the structure of these models grants them the property that for fixed values of variables v, the feasible-set polyhedron formed by variables z corresponds to one of a network flow model with integer extreme points. As a direct consequence, for each binary vector v there exists[1] an optimal vector z with only integer components. This is formalized next.

Proposition 4. *For fixed binary v, the set of feasible z in Models* (2.1) *and* (2.2) *describe a network flow polyhedron. Thus, for integer values of m, decision variables z will take integer values in an optimal solution.*

2.3.4 Incorporating lateness

In practical delivery problems, it is common that when a customer places an order, an *estimated time of arrival (ETA)* is announced and the operator seeks to serve the order no later than this time. In Models (2.1) and (2.2) we represent this idea by assuming that each order $j \in N$ must be served by Q_j (if served at all). In this section, we consider alternative models that allow some orders to be served if they arrive late. To do so, we now redefine Q_j to be the latest time that order j may be successfully served and introduce a new target delivery time q_j as the ETA by which order $j \in N$ is sought to be delivered. An order j

[1] This is true for Model (2.1), as long as the fixed value of m is integer and allows feasibility for the fixed v.

delivered at $t \in (q_j, Q_j]$ is then considered *late*.

Mathematically, let $s \in \{\tau_N, \tau_N + 1, \ldots, S\}$ be a target delivery guarantee for all orders. Then for each $j \in N$ we define $q_j \equiv r_j + s \leq Q_j$ as the target delivery time by which order $j \in N$ is desired to be delivered. From this definition we present a problem that seeks to minimize delivery lateness measured as the number of orders delivered after their target delivery time q_j.

Problem 3 (Late Orders Minimization)**.** *Given a fleet of couriers of size m, find a schedule for each courier that serves every order $j \in N$ by Q_j and such that the number of orders served later than q_j is minimized.*

For a given order $j \in N$, let $\mathcal{L}_j \equiv \{(t, \tau_j) \in \mathcal{V}_j : q_j + 1 \leq t \leq Q_j\}$ be the set of late service nodes of j. Then Problem 3 is solved by the following integer program.

$$\min \sum_{j \in N} \sum_{p \in \mathcal{L}_j} v_{jp} \tag{2.3a}$$

$$\text{s.t. (2.2b), (2.1c)} - \text{(2.1h)}$$

Objective (2.3a) minimizes the number of orders served later than the target service time q_j by penalizing the objective every time this occurs while all orders must be served by their due time Q_j. Note that Problem 3 is feasible if and only if the number of couriers m in the input is at least the optimal value of Problem 2, as otherwise Constraint (2.2b) will lead to infeasibility.

Note that Problem 3 could use an alternative lateness-based objective. For example, the decision maker may prefer to minimize the *total aggregated lateness* over all the orders, giving a larger penalty to orders that are served closer to their maximum acceptable delivery time Q_i. In our current formulation, this would only require replacing (2.3a) by the expression $\min \sum_{j \in N} \sum_{p \in \mathcal{L}_j} (t - q_j) v_{jp}$.

2.3.5 Radius management

In the earlier formulations, when determining the maximum number of orders that can be served by a fixed fleet of couriers the assumption was that the optimization model can selectively choose to provide or deny service to any individual order. Such a strategy is reasonable when determining an upper bound on maximum orders served in hindsight or with complete information. A potentially more realistic model for accepting or rejecting orders is to use a *service radius*: if an order is attempted to be placed at time t when the service radius is ρ, then the order must be served if $\tau \leq \rho$ and must be denied service otherwise.

In this section, we introduce modifications of the models to handle such radius-based order management decisions. In the basic model, we assume that a service radius is set at the beginning of the horizon and remains unchanged through the operating horizon. We formally state the decision problem as follows:

Problem 4 (Single Service Radius Maximization)**.** *Given a fleet of m couriers, find a schedule for each and a service radius ρ that maximize the number of served orders, where each order $j \in N$ is served if and only if $\tau_j \leq \rho$.*

From Problem 4 we can develop a natural extension that selects a (potentially different) service radius at R different fixed times $\{t_1, \ldots, t_R\} \subseteq \mathbf{T}$, where $t_1 \equiv 0$. For any $t \in \mathbf{T}$, let ρ_ℓ be the active radius during time interval $[t_\ell, t_{\ell+1})$, $\ell \in \{1, \ldots, R\}$ (with $t_{R+1} \equiv T$). Then for order $j \in N$ where $r_j \in [t_\ell, t_{\ell+1})$, j is served if and only if $\tau_j \leq \rho_\ell$. We mathematically formulate this extension as follows.

Problem 5 (Fixed-Time Radius Management Problem)**.** *Given a fleet of m couriers, find a schedule for each of them and service radii $\{\rho_\ell \in [0, \tau_N]\}_{\ell=1}^{R}$ that maximize the number of*

served orders, where if order $j \in N$ is such that $r_j \in [t_\ell, t_{\ell+1})$, then j is served if and only if $\tau_j \leq \rho_\ell$.

Note from the problem definition that if some ready time r_j coincides with a radius shifting time t_ℓ, we assume the radius adjustment is performed right before the order is placed.

Since $\{t_\ell\}_{\ell=1}^{R}$ are given, we can model Problem 5 by augmenting Model (2.1) with a few additional constraints.

Proposition 5. *For each $\ell \in \{1, \ldots, R\}$, let $B_\ell \subseteq N$ be a list of orders such that (i) $j \in B_\ell$ if and only if $r_j \in [t_\ell, t_{\ell+1})$; and (ii) elements of B_ℓ are sorted in ascending order of travel time from the depot to their delivery location, with $B_{\ell,i}$ denoting the i-th element of list B_ℓ (and so $\tau_{B_{\ell,i}} \leq \tau_{B_{\ell,i+1}}$). Then for solving Problem 5, it suffices to solve the integer program resulting from combining Model* (2.1) *with the extra linear constraints*

$$\sum_{p \in \mathcal{V}_{B_{\ell,i}}} v_{B_{\ell,i},p} \begin{cases} \geq \sum_{p \in \mathcal{V}_{B_{\ell,i+1}}} v_{B_{\ell,i+1},p} & \text{if } \tau_{B_{\ell,i}} < \tau_{B_{\ell,i+1}} \\ = \sum_{p \in \mathcal{V}_{B_{\ell,i+1}}} v_{B_{\ell,i+1},p} & \text{if } \tau_{B_{\ell,i}} = \tau_{B_{\ell,i+1}} \end{cases}, \quad \begin{array}{c} \forall \ell \in \{1, \ldots, R\} \\ \forall i \in \{1, \ldots, |B_\ell| - 1\} \end{array} \tag{2.4}$$

Moreover, given an optimal solution (v^*, z^*) of the resulting model, each optimal service radius can be recovered by computing

$$\rho_\ell^* = \max_{j \in B_\ell} \left\{ \tau_j : \sum_{p \in \mathcal{V}_j} v_{jp}^* = 1 \right\}, \quad \forall \ell \in \{1, \ldots, R\}$$

Adding Constraint set (2.4) forces an order to be served if the next furthest order placed from the depot during the same time interval ℓ is served while also forcing all orders during interval ℓ to be either served or not served if they have the same value of τ.

2.3.6 L-star extension

Although the setting considered up to now assumes that all the orders delivery locations lie in a single line segment with the depot at one of its extremes, our framework can easily be adapted to the more general case with an arbitrary L number of line segments radiating from the depot point. Note that this network topology assumes that all travel between line segments must transit the depot, and thus the only reasonable order bundles for dispatches are those where all orders are to be delivered in a common segment.

For $h \in \{1, \ldots, L\}$, let $N_h \subseteq N$ be the subset of N containing orders to be delivered in line segment h, with $n_h \equiv |N_h|$ and $\sum_{h=1}^{L} n_h = n$. Moreover, we assume that orders in each subset N_h are in ascending order of ready time. In addition, since $L \geq 1$ the delivery location of order $j \in N_h$ is now characterized by the pair (τ_j, h), representing a distance τ_j from the depot along the line segment h. As a result, nodes in the time-expanded network encode a time, a line segment and a distance from the depot. Defining $h = 0$ for nodes at the depot, we redefine depot nodes as $(t, 0, 0)$, and non-depot nodes as (t, τ_j, h).

Aside from these minor adjustments, the only procedure that requires a few additional considerations is the routine that creates the time-expanded networks. This is due to the dispatches at a returning time: at the time a courier returns from delivering orders at a particular line segment, it can either remain in the depot until the next order is ready or else be immediately dispatched into any of the L line segments with a new set of active orders. This implies that any depot node defined by the return of a courier defines an outbound arc to each of the L line segments containing the delivery location of an active order at that time. The adapted network building routine can be found in Appendix A.2.

Once the time-expanded network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ is created, consider the following redefinition of the sets of nodes: for $j \in N_h$, let $\mathcal{V}_j \equiv \{(t, \tau_j, h) \in \mathcal{V} : r_j + \tau_j \leq t \leq Q_j\}$ be the set of nodes where order j can be served. Similarly, let $\mathcal{V}_0 = \{(t, 0, 0) \in \mathcal{V}\}$. Lastly,

for each $p \in \mathcal{V}$ the sets of adjacent nodes δ_p^- and δ_p^+ are defined as in the previous section. With these modifications, the integer program formulations provided for problems from the previous section exactly model the corresponding L-star variant.

Once the time-expanded network $\mathcal{N}(\mathcal{V}, \mathcal{A})$ is built, modelling the k-star extensions of problems in previous sections. Indeed, it only remains to adapt the previous notation to this new setting. First consider the set of time stamps at which a depot node is defined by Algorithm 5, $\mathcal{T}_0 \equiv \bigcup_{l=1}^{k} \mathcal{T}_0^l$, and the collection of sets of time points $\{\mathcal{T}_j\}_{j \in N}$ at which a non-depot node is defined at delivery location (τ_j, l_j). Then for $j \in N$, let $\mathcal{V}_j \equiv \{(t; \tau_j, l_j) : t \in \mathcal{T}_j\}$ be the associated node set in $\mathcal{N}$. Similarly, let $\mathcal{V}_0 = \{(t; 0, 0) : t \in \mathcal{T}_0\}$, and $\mathcal{V} \equiv \bigcup_{j \in N \cup \{0\}} \mathcal{V}_j$. For each $p \in \mathcal{V}$, we define δ_p^- and δ_p^+ as in the previous section.

Considering this notation, it can be noted that the only difference between each of the models that consider a single line segment and its k-star extension is that in the latter, orders delivery locations are allowed to be at different sides of the depot. Therefore, changes in the formulation are limited to defining locations as a pair (τ, l) (in both the time-expanded network construction and in the decision variables definition), and the definition of proper nodes and arcs in the underlying time-expanded network (addressed by Algorithm 5).

2.4 Two-depot setting

2.4.1 Problem formulation

In Section 2.3 it is assumed that every order was placed to a single depot. In this section, we extend this setting to a single line segment with two depots, one located at each of its ends, that share a courier fleet to make deliveries. Customers place an order that is to be filled by a specific depot; for example, these locations may represent two different restaurants. Subject to some minor changes, we model and solve this case employing the same framework presented in Section 2.3.

Consider a line segment $[0, U]$ with $U > 0$, and two depots 1 and 2 located at $\tau_0 = 0$ and $\tau_U = U$, respectively. For depot $d \in \{1, 2\}$, let N_d be the set of orders that must be picked up from d, with $n_d \equiv |N_d|$ and $n \equiv n_1 + n_2$. Moreover, for each depot d we define

- A n_d-dimensional vector of ready times corresponding to orders placed at depot d, $\boldsymbol{r}^d \equiv (r_1^d, \ldots, r_{n_d}^d)$, whose components are sorted in increasing order. In the following, we label an order from depot d by j if the order has the j-th earliest ready time among the orders in such depot. Furthermore, without lost of generality, we assume $r_1^1 = 0$ and $r_1^2 \geq 0$.

- A corresponding n_d-dimensional vector of delivery locations $\boldsymbol{\tau}^d \equiv (\tau_1^d, \ldots, \tau_{n_d}^d)$, where the j-th component denotes the delivery location of order $j \in N_d$ *measured with respect to depot 1*.

- A corresponding n_d-dimensional vectors of target delivery times $\boldsymbol{q}^d \equiv (q_1^d, \ldots, q_{n_d}^d)$, and due times $\boldsymbol{Q}^d \equiv (Q_1^d, \ldots, Q_{n_d}^d)$.

Also, consider

$$\overline{d} \equiv \begin{cases} 2 & \text{if } d = 1 \\ 1 & \text{if } d = 2 \end{cases} \qquad \tilde{\tau}_j^d \equiv \begin{cases} \tau_j^d & \text{if } d = 1 \\ U - \tau_j^{\overline{d}} & \text{if } d = 2 \end{cases}$$

where $\overline{d}$ denotes the complement of depot d, and $\tilde{\tau}_j^d$ corresponds to the delivery location of order $j \in N_d$ measured from its depot d. Now we pose the two-depot version of the early problems as follows:

Problem 6 (Two-depot Order Maximization)**.** *Given depots 1 and 2, and a fleet of m identical couriers. Find a schedule for each courier that maximizes the total number of orders served in $N_1 \cup N_2$.*

Problem 7 (Two-depot Courier Minimization)**.** *Given two depots 1 and 2. Find the mini-*

mum number of identical couriers needed and a schedule for each of them, such that every order in $N_1 \cup N_2$ is served on time.

An important assumption made is that for all the considered two-depot problems, the decision maker has the ability to select at which depot each courier starts operating.

2.4.2 Time-expanded network construction

Next we present a routine that constructs a time-expanded network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$ with two depots. This new algorithm creates a time-expanded sub-network for each of the two depots (with their corresponding depot and non-depot nodes) that are connected through depot nodes. In order to make the distinction between both sub-networks explicit we redefine every node in the network as a tuple $v = (t, s, d)$, where s represents a spatial location, t a time point, and $d \in \{1, 2\}$ an associated sub-network based on a corresponding depot. Similarly, every arc (v_1, v_2) is now associated to a sub-network which is given by the sub-network of v_1. For the network construction routine, see Appendix A.3.

2.4.3 Integer program formulations

It is not hard to extend models from Section 2.3 to formulate Problems 6 and 7. Indeed, it suffices to redefine the decision variables and constraints used in the single depot models but incorporating into the existing notation the depot $d \in \{1, 2\}$ sub-network to which each node belongs to. First consider the sets of time stamps at which depot nodes for each depot are defined by Algorithm 7, namely $\mathcal{T}_0$ and $\mathcal{T}_U$ for depots 1 and 2, respectively. For depot $d \in \{1, 2\}$ and $j \in N_d$, let $\mathcal{V}_j^d \equiv \{(t, \tau_j, d) \in \mathcal{V} : r_j^d + \tilde{\tau}_j^d \leq t \leq Q_j^d\}$ be the nodes set where order j can be served. Similarly, let $\mathcal{V}_0 = \{(t, 0, 1) \in \mathcal{V}\}$ and $\mathcal{V}_U = \{(t, U, 2) \in \mathcal{V}\}$ the sets of depot nodes at depot 1 and 2, respectively. Lastly, for each node $p \in \mathcal{V}$ consider the sets of adjacent nodes δ_p^- and δ_p^+ defined as in past sections. Then consider the following

decision variables:

$$z_{pq} = \text{Number of couriers that traverse arc } (p,q) \in \mathcal{A}$$

$$v_{jp}^d = \begin{cases} 1 & \text{if order } j \in N_d \text{ is served at node } p \in \mathcal{V}_j^d,\ d \in \{1,2\} \\ 0 & \text{otherwise} \end{cases}$$

We can formulate Problem 6 as the following integer program.

$$\max \sum_{d=1}^{2} \sum_{j \in N_d} \sum_{p \in \mathcal{V}_j^d} v_{jp}^d \tag{2.5a}$$

$$\text{s.t.} \sum_{t \in \mathcal{T}_j} v_{jt}^d \leq 1, \quad \forall d \in \{1,2\}, \quad \forall j \in N_d \tag{2.5b}$$

$$v_{jp}^d \leq \sum_{q \in \delta_p^-} z_{qp}, \quad \forall d \in \{1,2\}, \quad \forall j \in N_d, \quad \forall p \in \mathcal{V}_j^d \tag{2.5c}$$

$$\sum_{q \in \delta_{(0,0,1)}^+} z_{(0,0,1),q} = m \tag{2.5d}$$

$$\sum_{p \in \delta_{(T,0,1)}^-} z_{p,(T,0,1)} = m \tag{2.5e}$$

$$\sum_{p \in \delta_q^-} z_{pq} = \sum_{r \in \delta_q^+} z_{qr}, \quad \forall q \in \mathcal{V} \setminus \{(0,0,1),(T,0,1)\} \tag{2.5f}$$

$$v_{jp}^d \in \{0,1\}, \quad \forall d \in \{1,2\}, \quad \forall j \in N_d, \quad \forall p \in \mathcal{V}_j^d \tag{2.5g}$$

$$z_{pq} \in \begin{cases} \mathbb{R}_+ & \text{if } p,q \in \mathcal{V}_0 \cup \mathcal{V}_U \\ \{0,1\} & \text{otherwise} \end{cases}, \quad \forall (p,q) \in \mathcal{A} \tag{2.5h}$$

Model (2.5) is very similar to Model (2.1). Objective (2.5a) seeks to maximize the total number of served orders. For each depot $d \in \{1,2\}$, Constraint (2.5b) enforces that each order $j \in N_d$ is served at most once. Constraint (2.5c) requires a courier at node $(t, \tau_j, d) \in \mathcal{V}_j^d$ for order $j \in N_d$ to be served at time t. Constraints (2.5d) - (2.5f) are courier flow constraints; note that the insertion of arc $((0,0,1),(r_1^2,U,2))$ allows the model

to select how many of the m couriers start the operating horizon at each depot, and arc $((T, U, 2), (T, 0, 1))$ leaves $(T, 0, 1)$ as the unique sink in network $\mathcal{N}$. Lastly, constraints (2.5g) and (2.5h) specify the domain of the decision variables.

Similarly, a formulation for Problem 7 can be obtained by extending Model (2.2) as follows:

$$\min \quad m \tag{2.6a}$$

$$\text{s.t.} \quad \sum_{p \in \mathcal{V}_j^d} v_{jp}^d = 1, \quad \forall d \in \{1, 2\}, \quad \forall j \in N_d \tag{2.6b}$$

$$(2.5c) - (2.5h)$$

$$m \in \mathbb{R}_+ \tag{2.6c}$$

The similarities between formulations for the single depot and two-depot cases allow to preserve the result in Proposition 4 for two-depot models.

2.4.4 Incorporating lateness

Now we introduce the notion of lateness for two-depot models by incorporating the target delivery time q_j^d.

Problem 8 (Two-depot Late Orders Minimization)**.** *Given depots 1 and 2 and a fleet of m identical couriers. Find a schedule for each courier such that every order $j \in N_d$, $d \in \{1, 2\}$ is served by its due time Q_j^d and such that the number of orders served after the target delivery time q_j^d is minimized.*

For $d \in \{1, 2\}$ and $j \in N_d$, let $\mathcal{L}_j^d \equiv \{(\tau_j, t, d) \in \mathcal{V}_j^d : q_j^d + 1 \leq t \leq Q_j^d\}$. Then Problem 8

is formulated as

$$\min \sum_{d=1}^{2} \sum_{j \in N_d} \sum_{p \in \mathcal{L}_j^d} v_{jp}^d \tag{2.7a}$$

$$\text{s.t. (2.6b), (2.5c)} - \text{(2.5h)}$$

Model (2.7) minimizes the number of orders that are served after the corresponding target delivery q_j^d subject to every orders being served by its due time Q_j^d (Constraint (2.6b)) and courier flow constraints (2.5c) - (2.5h). Feasibility of Problem 8 is equivalent to the input number of couriers m being at least the optimal value of Problem 7, as otherwise Constraint (2.6b) cannot be satisfied.

2.4.5 Radius management

We adapt the radius management models from Section 2.3 to the two-depot case. In this setting, each depot $d \in \{1, 2\}$ may select a service radius up to $R_d \geq 1$ times during the operating horizon, with the first radius being selected at $t_1^d \equiv r_j^d$, and any order whose delivery location lies inside the active radius at the moment of its placement must be served. Mathematically, let $\{t_\ell^d\}_{\ell=1}^{R_d}$ be the times at which depot d changes its radius, and let ρ_ℓ^d be the radius selected at time t_ℓ^d. If order $j \in N_d$ satisfies $r_j^d \in [t_\ell^d, t_{\ell+1}^d)$ (with $t_{R_d+1}^d \equiv T$), then j is served if and only if and $\tilde{\tau}_j^d \leq \rho_\ell^d$.

Now we present the two-depot version of the problem:

Problem 9 (Two-depot Fixed-Time Service Radius Management Problem)**.** *Given depots 1 and 2 and a fleet of m identical couriers. Find a schedule for each courier and service radii $\{\rho_\ell^d \in [0, \max_{i \in N_d}\{\tilde{\tau}_i^d\}]\}_{\ell=1}^{R_d}$ for depots $d \in \{1, 2\}$ such that the total number of served orders is maximized, where if some order $j \in N_d$ is such that $r_j^d \in [t_\ell^d, t_{\ell+1}^d)$, then such order is served if and only if $\tilde{\tau}_j^d \leq \rho_\ell^d$.*

In order to solve Problem 9, it is enough to add a small set of linear constraints to Model (2.5):

Proposition 6. *Let $\{B_\ell^d\}_{\ell=1}^{R_d}$ be the list of orders j such that $r_j^d \in [t_\ell^d, t_{\ell+1}^d)$ for depot $d \in \{1,2\}$, obtained from Algorithm 4 by replacing τ_j^d in its input by $\tilde{\tau}_j{}^d$; and let $B_{\ell,i}^d$ be the i-th element of list B_ℓ^d. Then Problem 9 can be formulated by adding the following linear constraints to Model* (2.5)*:*

$$\sum_{p \in \mathcal{V}^d_{B^d_{\ell,i}}} v^d_{B^d_{\ell,i},p} \begin{cases} \geq \sum_{p \in \mathcal{V}^d_{B^d_{\ell,i+1}}} v_{B^d_{\ell,i+1},p} & \text{if } \tilde{\tau}^d_{B^d_{\ell,i}} < \tilde{\tau}^d_{B^d_{\ell,i+1}} \\ = \sum_{p \in \mathcal{V}^d_{B^d_{\ell,i+1}}} v_{B^d_{\ell,i+1},p} & \text{if } \tilde{\tau}_{B^d_{\ell,i}} = \tilde{\tau}_{B^d_{\ell,i+1}} \end{cases}, \qquad \begin{array}{c} \forall d \in \{1,2\} \\ \forall \ell \in \{1,\ldots,R_d\} \\ \forall i \in \{1,\ldots,|B_\ell^d|-1\} \end{array} \tag{2.8}$$

Moreover, given an optimal solution (v^{1*}, v^{2*}, z^*) for the resulting model, the optimal service radii can be determined as

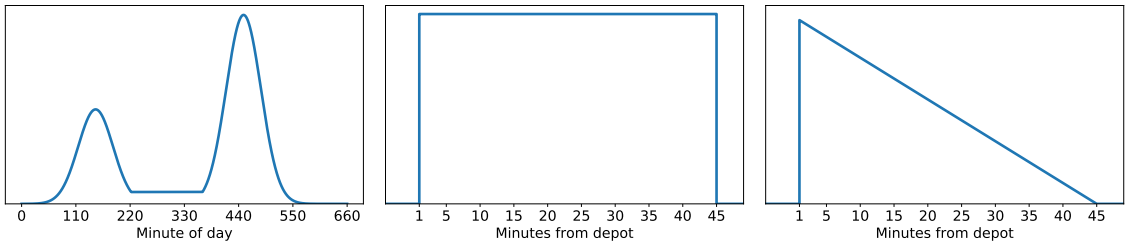
$$\rho_\ell^{d*} = \max_{j \in B_\ell} \left\{ \tilde{\tau}_j^d : \sum_{p \in \mathcal{V}_j^p} v_{jp}^{d*} = 1 \right\}, \quad \forall d \in \{1,2\}, \quad \forall \ell \in \{1,\ldots,R_d\}$$

2.5 Computational study

In this section we report results from solving the proposed models on various instances to gain insights about required fleet sizes and demand management strategies in meal delivery systems. The conducted experiments are separated into three subsections which, respectively, provide understanding about (i) the minimum fleet sizes required to serve delivery requests for various instances, (ii) the value of establishing an individual target delivery time to manage delivery lateness, and (iii) the potential benefits of dynamically adjusting a depot coverage radius as a demand management strategy.

Tested values for parameters n, S, m and s vary with the type of experiment and are shown at the beginning of each experiment subsection. The remaining parameters are either assumed constant or defined as a function of the aforementioned ones. In particular:

- All the instances consider a time horizon length of $T = 660$ minutes (11 hours).
- Order ready times r_i^d are obtained by randomly sampling from a continuous bimodal distribution and rounding each element to its nearest integer. This sampling distribution is an attempt to model realistic meal delivery operations, where it is observed that meal delivery demand is highly concentrated at lunch and dinner times, as illustrated in Figure 2.2a.
- Travel times τ_j^d are obtained by first drawing a random number from a continuous distribution and then rounding each element to its nearest integer (minute). The distribution we use depends on the number of depots considered:
  - For single depot settings we use a uniform distribution and a triangular distribution to sample delivery locations. These are illustrated in Figures 2.2b and 2.2c, respectively.
  - For the two-depot scenario we test two different levels of separation, $U \in \{60, 90\}$ minutes, each with its own sampling scheme. For $U = 60$ travel time to delivery locations of orders from depot 1 and 2 are sampled from triangular distributions $Tria(1, 1, 45)$ and $Tria(15, 59, 59)$, respectively. On the other hand, for $U = 90$ travel times to delivery locations of orders from depot 1 and 2 are generated from triangular distributions $Tria(1, 1, 45)$ and $Tria(45, 89, 89)$, respectively.
- All single depot instance settings consider $L = 4$ number of line segments. Moreover, orders are assumed to be distributed between the L line segments in such a way



(a) Bimodal distribution of r_j (b) $\tau_j \sim Unif(1, 45)$ (c) $\tau_j \sim Tria(1, 1, 45)$

Figure 2.2: Sample distributions

Table 2.2: Parameter values used for fleet size minimization experiments

Setting	n	S
Single depot	$\{75, 100, 120, 150\}$	$\{45, 50, 55, 60\}$
Two depots	$\{150, 200, 240, 300\}$	$\{45, 50, 55, 60\}$

that for any two line segments, the numbers of orders to be delivered in each never differ by more than one order.

- For the two-depot scenario we further consider that each order is assigned to a specific depot at random with equal probability.

2.5.1 Minimum fleet size

In this section we solve problems that determine the minimum courier fleet size required to serve all orders in a specific instance. Parameter values considered for this section are summarized in Table 2.2. We run 50 replications for each possible combination of settings and parameter values and report statistics on optimal courier fleet size m^*, and on bundle size, namely the number of orders in a single dispatch. Additionally, for the two-depot setting we analyze the number of crosses between both depots, proposing an auxiliary integer program that is able to show exactly when fleet sharing between depots yields a better operational cost than having each depot delivering orders with its exclusive fleet.

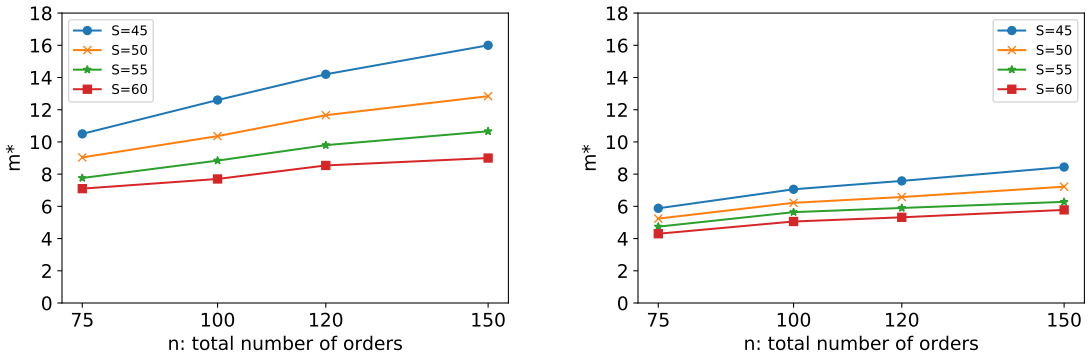
Single depot setting. Table 2.3 presents the obtained average fleet size values m^* for some of the considered single depot instances, and Figure 2.3 illustrates the effect of the

Table 2.3: Average m^* for some single depot configurations

(n, S)	$\tau_j \sim Unif(1, 45)$	$\tau_j \sim Tria(1, 1, 45)$	Average
$(75, 45)$	10.50	5.88	8.19
$(150, 45)$	16.00	8.44	12.22
$(75, 60)$	7.10	4.30	5.70
$(150, 60)$	9.00	5.78	7.39
Average	10.65	6.10	

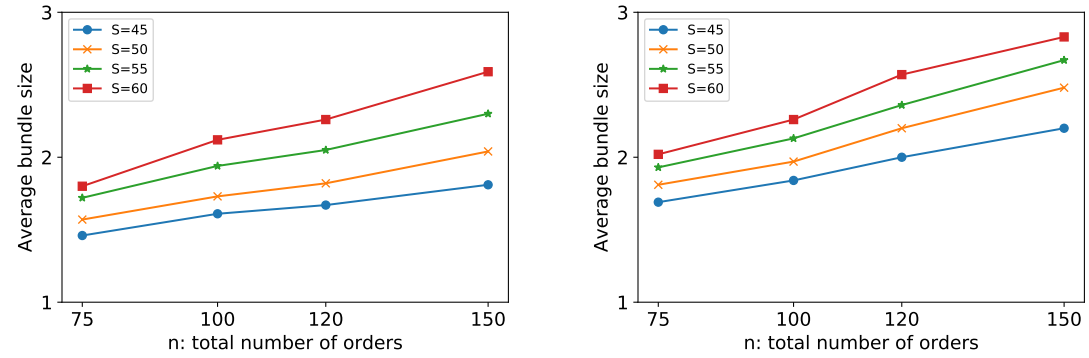
different parameters involved on the optimal fleet size. As expected, m^* increases as either n increases, S decreases, or order delivery locations become more distant to the depot. However, these effects on m^* differ in magnitude. To illustrate this, consider the scenario $(n, S) = (75, 60)$ as a base case, which corresponds to the case with fewest orders and most flexible due times. Observe that everything else equal, doubling the number of orders to $n = 150$ leads to a 30% increase in m^*; on the other hand, decreasing S by 25% to $S = 45$ while keeping all other values constant requires a relatively higher 44% increase in fleet size.

This difference in the effect on m^* is explained by both the bundling of multiple orders in a single dispatch and by the reduction in the dwell time of a courier who has returned to the depot before leaving for a subsequent dispatch. Since couriers are modeled assuming no limit on bundled orders, increasing the number of orders n tends to increase the number of orders bundled per dispatch and reduce the courier dwell time at the depot thus increasing courier productivity; the fleet size growth is modest with n. However, increasing the tightness of the delivery windows by decreasing S makes bundling orders less likely overall thus resulting in faster growth in required fleet sizes when n grows for smaller S. We note that, although bundle sizes are not constrained, the average numbers of orders bundled together for delivery lies between 1.5 and 3 for all instances; these averages are illustrated in Figure 2.4.



(a) $\tau_j \sim Unif(1, 45)$ (b) $\tau_j \sim Tria(1, 1, 45)$

Figure 2.3: Average m^* for a single depot



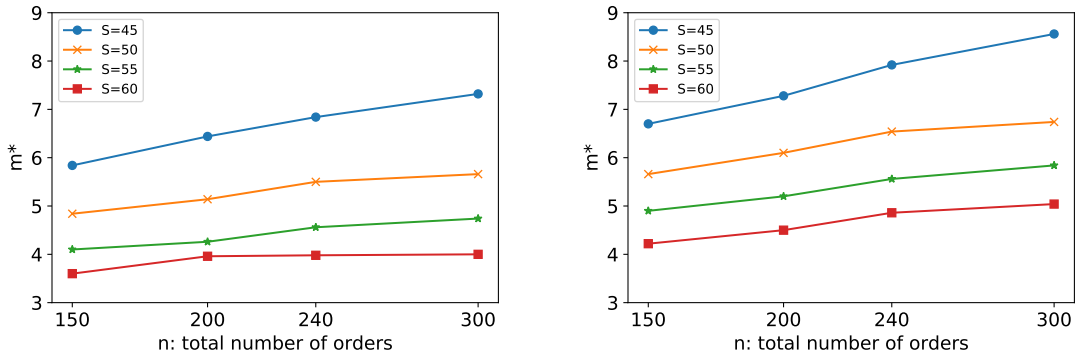
(a) $\tau_j \sim Unif(1, 45)$ (b) $\tau_j \sim Tria(1, 1, 45)$

Figure 2.4: Average bundle size for a single depot

The customer location distribution also plays an important role in determining the optimal fleet size. Indeed, switching the location distribution from triangular to uniform results in an average 75% increase in the required minimum fleet size; we note that this change substantially increases the average distance from the depot to a customer and concomitantly the duration of any given delivery dispatch.

Two-depot setting. Results for the two-depot minimum fleet sizes m^* and how they vary with n, S and U are depicted by Figure 2.5, and partial results are reported in Table 2.4. As in the single depot case, consider the base case $(n, S) = (150, 60)$ which has the least

number of orders and the most flexible delivery due times. Note that doubling n while preserving S results in an average 15.6% larger m^*, whereas only decreasing S to 45 minutes leads to a substantial increase of 60.4% in the required fleet size. Again, the ability to build larger bundles with larger values of S is critical to keeping fleet sizes from growing too large.



(a) $U = 60$ (b) $U = 90$

Figure 2.5: Average m^* for two depots

Taking a closer look at the level of separation between depots, we observe that increasing U from 60 to 90 requires an 18% larger average fleet size. This difference is explained by the potential benefit for sharing couriers in the fleet between depots and this benefit diminishes when the time required to transfer from one depot to another (after serving a final customer) grows large when compared to the time required to return to the original depot. Table 2.4 summarizes specific minimum fleet sizes for some representative values of n and S, and can be compared directly to the results in Table 2.5 that compute minimum fleet sizes when dedicated fleets are used at each depot. Our findings from this comparison show that without fleet sharing, the system would require a 21% larger number of couriers compared to the shared fleet size for $U = 60$. On the other hand, for a larger separation of $U = 90$ the benefit from fleet sharing is only 2.4%.

In terms of bundling, the average bundle size and its relationship with n, S and U are

Table 2.4: Average m^* for two depots

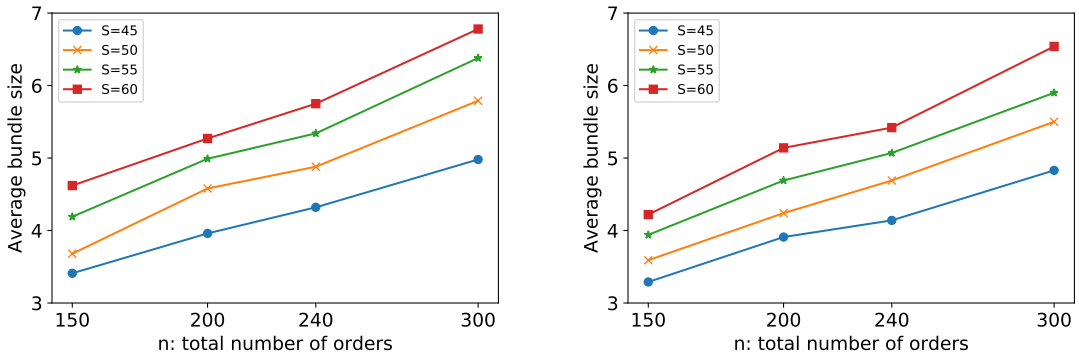
(n, S)	$U = 60$	$U = 90$	Average
$(150, 45)$	5.84	6.70	6.27
$(300, 45)$	7.32	8.56	7.94
$(150, 60)$	3.60	4.22	3.91
$(300, 60)$	4.00	5.04	4.52
Average	5.19	6.13	

Table 2.5: Average m^* for two depots (without fleet sharing)

(n, S)	Average m^*
$(150, 45)$	6.77
$(300, 45)$	8.77
$(150, 60)$	4.25
$(300, 60)$	5.33
Average	6.28

presented in Figure 2.6. As previously mentioned, we observe that the average bundle size follows a similar pattern with respect to n and S as the one observed for a single depot. We also see that the average bundle size is slightly larger comparatively for the $U = 60$ instances versus the $U = 90$ instances and this is consistent with the smaller fleet sizes required for the former. Finally, it should also be noted that the bundle sizes in these two depot instances are roughly twice the size of those for single depot instances. This is due to the fact that the same number of total orders are distributed over two line segments (one from depot 1 and the other from depot 2) in these instances and distributed over four line segments in the single depot instances, so the order density per time is effectively doubled.

It is also interesting to analyze how often couriers cross the line segment from one depot to the other in these two-depot instances. To do so, we solve a second integer program for each instance that, for a given optimal fleet size m^*, computes the minimum number of crosses between depots required to serve all the orders feasibly. Let $\mathcal{A}^*$ be the set of arcs that traverse from a non-depot node to a depot node such that both nodes are associated with different depot sub-networks, and let m^* be the optimal fleet size obtained from solving



(a) $U = 60$ (b) $U = 90$

Figure 2.6: Average bundle size for two depots

Problem 7. Then the integer program is formulated as follows:

$$\min \sum_{(p,q)\in\mathcal{A}^*} z_{pq} \tag{2.9a}$$

$$\text{s.t.} \quad (2.6b), (2.5c), (2.5f) - (2.5h)$$

$$\sum_{q\in\delta^+_{(0,0,1)}} z_{(0,0,1),q} = m^* \tag{2.9b}$$

$$\sum_{p\in\delta^-_{(T,0,1)}} z_{p,(T,0,1)} = m^* \tag{2.9c}$$

Objective (2.9a) minimizes the number of courier crosses between depots. In addition, we replace the decision variable m by the optimal fleet size m^* in constraints (2.9b) and (2.9c).

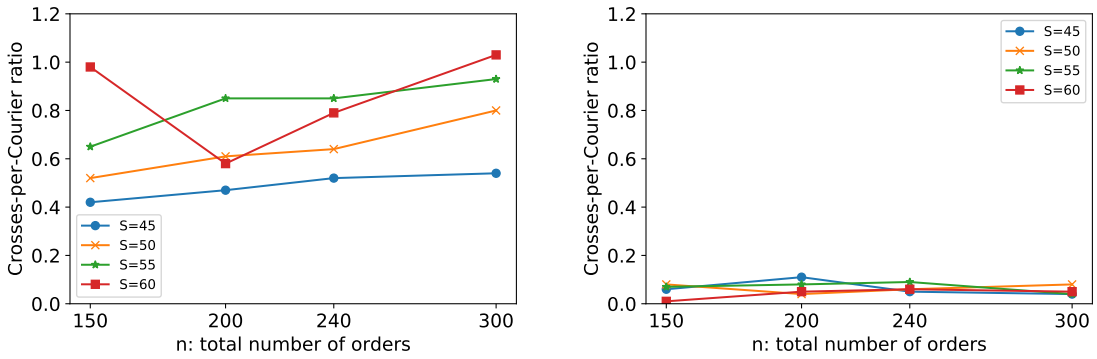
Due to the optimality of m^*, a key property of this auxiliary integer program is that it yields an optimal value of 0 crosses if and only if solving the two-depot instance with fleet sharing does not give any savings in the number of couriers with respect to employing individual fleets. Therefore, a strictly positive number of crosses reveals potential benefits from allowing a shared fleet for both depots. Table 2.6 shows the percentage of solved instances for which fleet sharing results in strictly fewer couriers than using dedicated courier fleets for each of the depots. Almost all instances yield a strictly positive minimum number of

Table 2.6: Percentage of instances whose optimal number of crossings is strictly positive

(n, S)	$U = 60$	$U = 90$
(150, 45)	84%	28%
(150, 60)	62%	4%
(300, 45)	96%	28%
(300, 60)	90%	22%

crossings for $U = 60$, but when $U = 90$ the benefit is much more limited.

In order to control for the possible impact of the fleet size m^* on the optimal number of crosses, consider the ratio between the number of crosses and m^*, which we denote by γ and depict in Figure 2.7. The effect of the level of separation U on γ is evident: the operation exploits the short inter-depot traveling times when $U = 60$ by dynamically reallocating couriers between depots. On the other hand, the larger value $U = 90$ leads to inter-depot traveling that is too time consuming to be effective, and therefore the number of crosses per courier is usually below one in ten.



(a) $U = 60$ (b) $U = 90$

Figure 2.7: Average γ for different instance settings

Interestingly, when $U = 60$ we observe that in general, γ is non-decreasing in both S and n, with the exception of the case $S = 60$. In particular, we observe a significant decrease in γ when orders are increased from 150 to 200 which also corresponds to a significant increase in the minimum fleet size m^*. As n is further increased, the fleet size does not increase and

Table 2.7: Parameter values used for lateness experiments

Setting	n	S	s
Single depot	$\{75, 150\}$	$\{60\}$	$\{45, 50, 55\}$
Two depots	$\{150, 300\}$	$\{60\}$	$\{45, 50, 55\}$

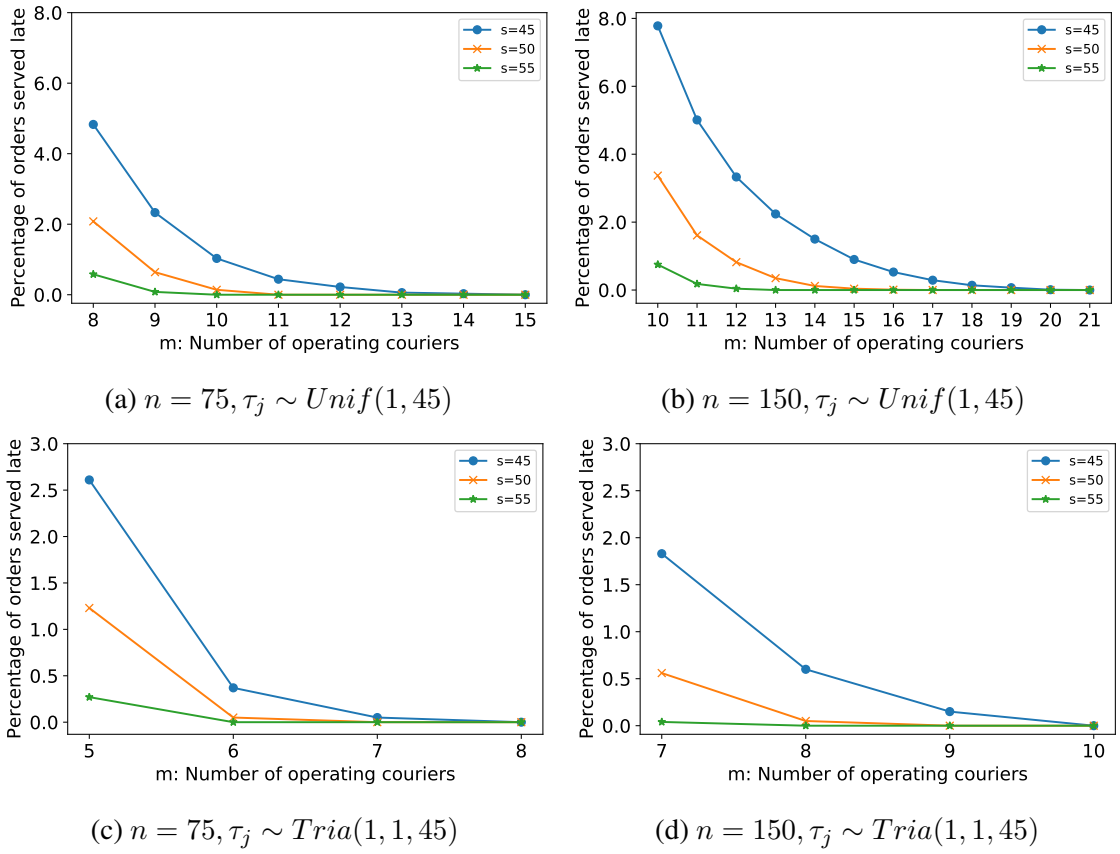
more orders are handled by the same number of couriers, many of which execute crosses from one depot to the other. Hence, more crosses-per-courier are observed.

2.5.2 Analysis of lateness via target delivery time

Now we report findings from solving problems that minimize the number of late orders. The results in this section demonstrate the benefit of introducing a target due time $q_j = r_j + s$ as a simple approach for balancing the flexibility of the system between setting too large and too restrictive due times $Q_j = r_j + S$. We empirically show that combining a restrictive target service level s with a flexible service level S can be effective, leading to solutions with very few late orders that use significantly fewer couriers than more restrictive settings with tighter values of S. The experiments considered in this section were conducted using the parameter values listed in Table 2.7. Note that the selected value of S used in this section corresponds to its most flexible value in Section 2.5.1, whereas the range of values of s begins with the most restrictive value. For each combination of parameters (n, S, s, U) we randomly generate 50 stream of orders and compute the minimum fraction of orders delivered after their target due time q_j for different number of available couriers m. To preserve feasibility, we only consider fleet sizes no less than the minimum number of couriers required to serve all orders by their due time Q_j.

Single depot setting. The results for this section are presented in Figure 2.8. We observe that the delivery location proximity to the depot has a considerable effect on the fleet size required to maintain a given level of lateness. Indeed, for some values of (n, S, s), maintaining a given percentage of late orders requires a fleet size that can be over 100% larger for the uniformly-distributed locations case when compared to the triangular distribution

locations.



(a) $n = 75, \tau_j \sim Unif(1, 45)$

(b) $n = 150, \tau_j \sim Unif(1, 45)$

(c) $n = 75, \tau_j \sim Tria(1, 1, 45)$

(d) $n = 150, \tau_j \sim Tria(1, 1, 45)$

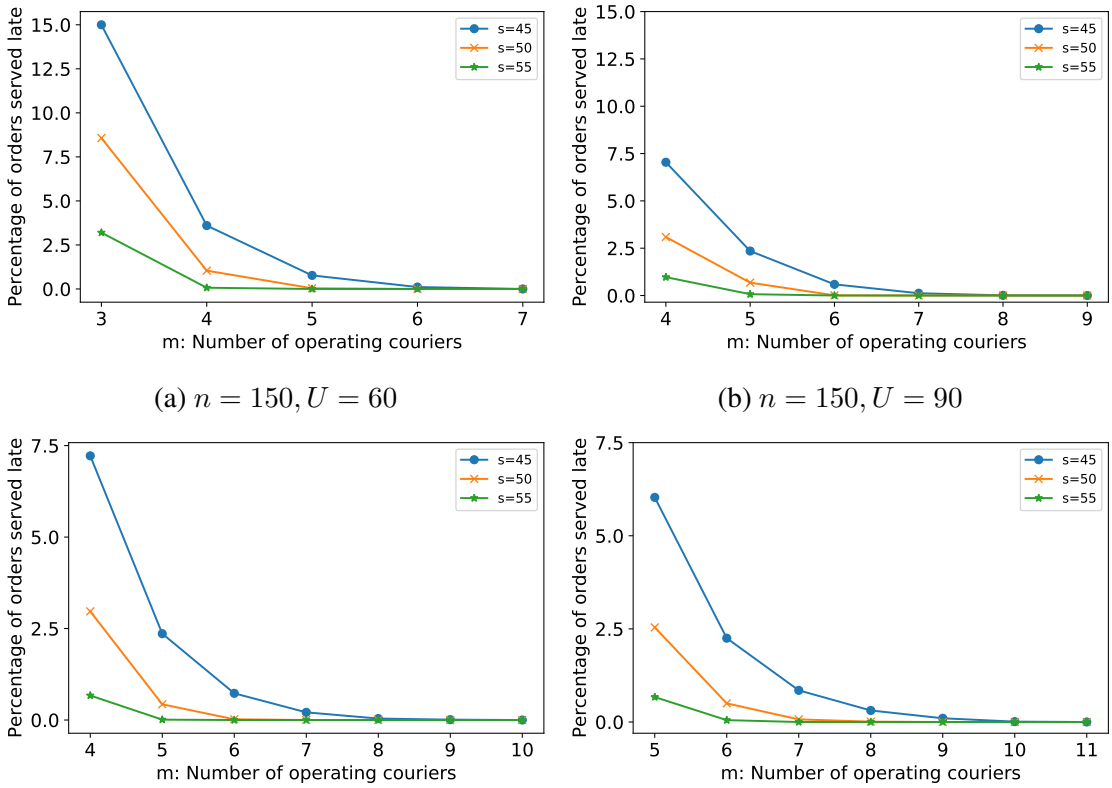
Figure 2.8: Average fraction of late orders for a single depot

As expected, our findings show that the most critical factor for determining the fraction of late orders is s. A small enough value of this parameter strongly restricts the flexibility of the system, leading to a substantial increase in the fraction of late orders for smaller fleet sizes. This is caused since lower values of s restrict the bundling opportunities for orders to be delivered before the target delivery time.

Lastly, we compare the setting that considers both target and hard due times q_j and Q_j against the case that only includes due times Q_j. Note that only considering a hard due time corresponds to the particular scenario of having a target due time that satisfies $s = S$. To illustrate the benefits from having a target due time, consider the instances with $n = 150$ orders where delivery locations are uniformly distributed. Note that if $S = s = 45$ minutes,

then on average about $m = 16$ couriers are needed to achieve on-time service. However, if the maximum delivery time S is relaxed to 60 minutes while maintaining $s = 45$, a 33% smaller courier fleet still manages to serve all 150 orders with only 5% of them delivered late (after q_j). Of course, even fewer couriers would be required by setting $S = s = 60$ minutes but the average time to delivery of the orders would increase.

Two-depot setting. The effects from s and n on the fraction of late orders in instances with two depots are similar to those observed for a single depot and are shown in Figure 2.9. We see that varying U has a significant effect on the number of late orders. For small fleet sizes for which flexibility is limited, increasing U from 60 to 90 can on average scale up the fraction of late orders by a factor of 2.



(a) $n = 150, U = 60$

(b) $n = 150, U = 90$

(c) $n = 300, U = 60$

(d) $n = 300, U = 90$

Figure 2.9: Average fraction of late orders for two depots

Additionally, we again observe some advantages from considering both a target delivery

time and a hard due time. As shown in Figure 2.9 for $n = 300$ orders and a time between depots of $U = 60$, the simple case $S = s = 45$ results in a conservative solution that requires a total of $m = 8$ couriers in order to achieve full service with no late orders on average. Alternatively, relaxing S to 60 minutes while keeping $s = 45$ offers a reasonably more flexible solution that is able to serve all orders with an approximately 30% smaller fleet with only 2.5% of orders served late. Similar results can be observed for the remaining (n, U) pairs: a substantial reduction of 33% of the fleet that serves all orders without lateness results in a mild increase of late orders of less than 5%.

2.5.3 Demand management via service radius adjustments

In this section we study *order demand management* using our modeling framework. Specifically, we consider demand management strategies that are driven by a selected *service radius* from the depot (from which the order is placed). Radius-based strategies are simple: once a radius length ρ is selected, deliveries must be made to any order placed by a customer with a delivery location τ within the disk around the depot with radius ρ; in our simple geometric settings, this corresponds to $\tau \leq \rho$. We will compare the performance of demand management strategies when the service radius is selected and fixed in advance to those where the radius may change during the operating day using Problems 5 and 9.

Tables 2.8 and 2.9 summarize the parameters used in this section, where R_d measures the number of times the service radius is adjusted during the operating day. The case $R_d = 1$ is referred to as the *base case* and consists of simply setting a unique radius beginning at time $t = 0$. On the other hand, the case $R_d = \infty$ is referred to as the *selective-service case* and, since the radius can change at every order ready time, is equivalent to the settings of Problems 1 and 6 where the operator selectively chooses to either accept or reject each order; while this case is unrealistic in practice, it provides an upper bound on system performance. To measure the effectiveness of radius-based demand management, we experiment with 50 randomly-generated order streams and focus primarily on the fraction of

Table 2.8: Parameter values used for demand management experiments

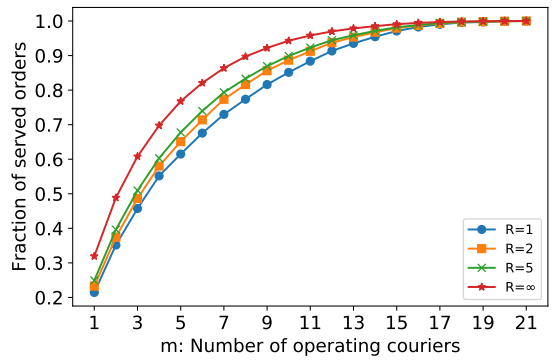
Setting	n	S	R_d
Single depot	$\{75, 150\}$	$\{45, 60\}$	$\{1, 2, 5, \infty\}$
Two depots	$\{75, 150\}$	$\{45, 60\}$	$\{1, 2, 5, \infty\}$

Table 2.9: Times at which the service radius may change

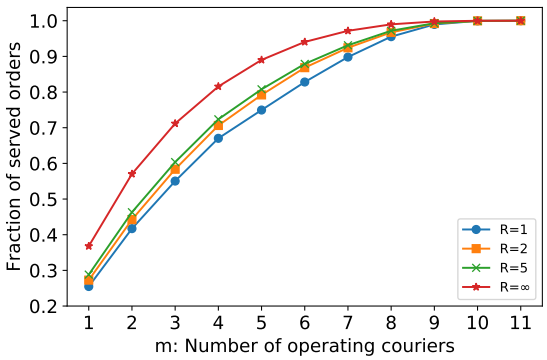
R_d	1	2	5	∞
$\{t_l^d\}_{l=1}^{R_d}$	$\{0\}$	$\{0, 300\}$	$\{0, 100, 200, 400, 500\}$	$\{r_j^d\}_{j \in N_d}$

the n orders served as a function of the amount of available resources.

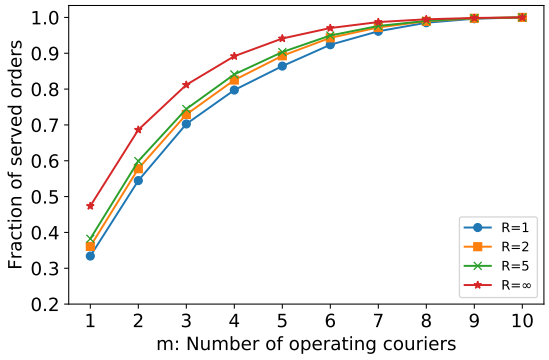
Single depot setting. Figure 2.10 shows the fraction of served orders in optimal solutions to Problem 5 for different values of R when $n = 150$; the subfigures provide results for different combinations of S and the travel time distribution. Problems were solved for fleet sizes m from one to the minimum fleet size required to serve all orders in all instances.



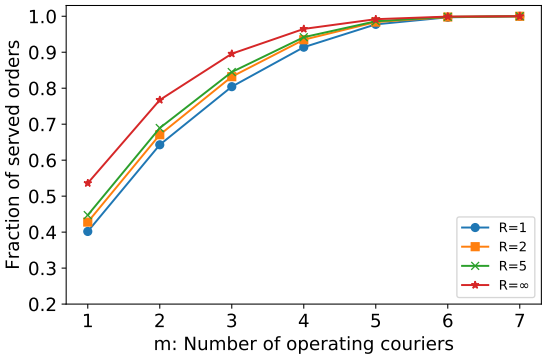
(a) $S = 45, \tau_j \sim Unif(1, 45)$



(b) $S = 60, \tau_j \sim Unif(1, 45)$



(c) $S = 45, \tau_j \sim Tria(1, 1, 45)$



(d) $S = 60, \tau_j \sim Tria(1, 1, 45)$

Figure 2.10: Fraction of served orders with $n = 150$ for a single depot

Although increasing R results in more flexibility for the operator to decide when and where to accept orders, the value of this flexibility depends on the available fleet size. In general, the results indicate that the largest benefit of increasing R occurs for instances with fleets of medium size (not too small and not too large). In such cases, increasing R from $R = 1$ to $R = 2$ can on average close approximately 30% of the gap to the upper bound; increasing to $R = 5$ closes the gap by approximately 50%. Systems with larger fleets intuitively benefit less from increased values of R. However, we also see that the smallest fleets that can only serve a small fraction of the orders do not benefit substantially from small increases in R; larger jumps in the fraction of orders served only occur when individual orders can be accepted or rejected (as in the $R = \infty$ case).

The results also help us understand the potential fleet size savings that can be achieved when the objective is to serve some fixed fraction of potential orders as R is increased. For example, when $n = 150$, $S = 45$, and the order location distribution is uniform, 21 couriers are required to serve all orders. However, using a radius-based demand management scheme with $R = 1$ leads to a fleet size requirement of 14 couriers to serve 95% of all orders. This fleet can be reduced again to 13 couriers when $R = 2$. Table 2.10 summarizes the results for a large set of scenarios and show that significantly smaller fleets can lead to reasonable order coverage fractions. We also see that it is typical when we require 80% or 95% demand coverage that when $R = 5$, the number of couriers required is often either the same or just one more than the fleet required in the selective-service $R = \infty$ case.

Lastly, we observe that S and the distribution of τ_j lead to large variations in the fleet size required to achieve a certain demand coverage fraction. In particular, decreasing S from 60 to 45 minutes may result in an average increase in the fleet size ranging between 30% and 60% to maintain a fixed level of service, while changes in the delivery location distribution from $Tria(1, 1, 45)$ to $Unif(1, 45)$ may require even doubling the courier fleet.

Two-depot setting. When solving instances with two depots, we found that our proposed

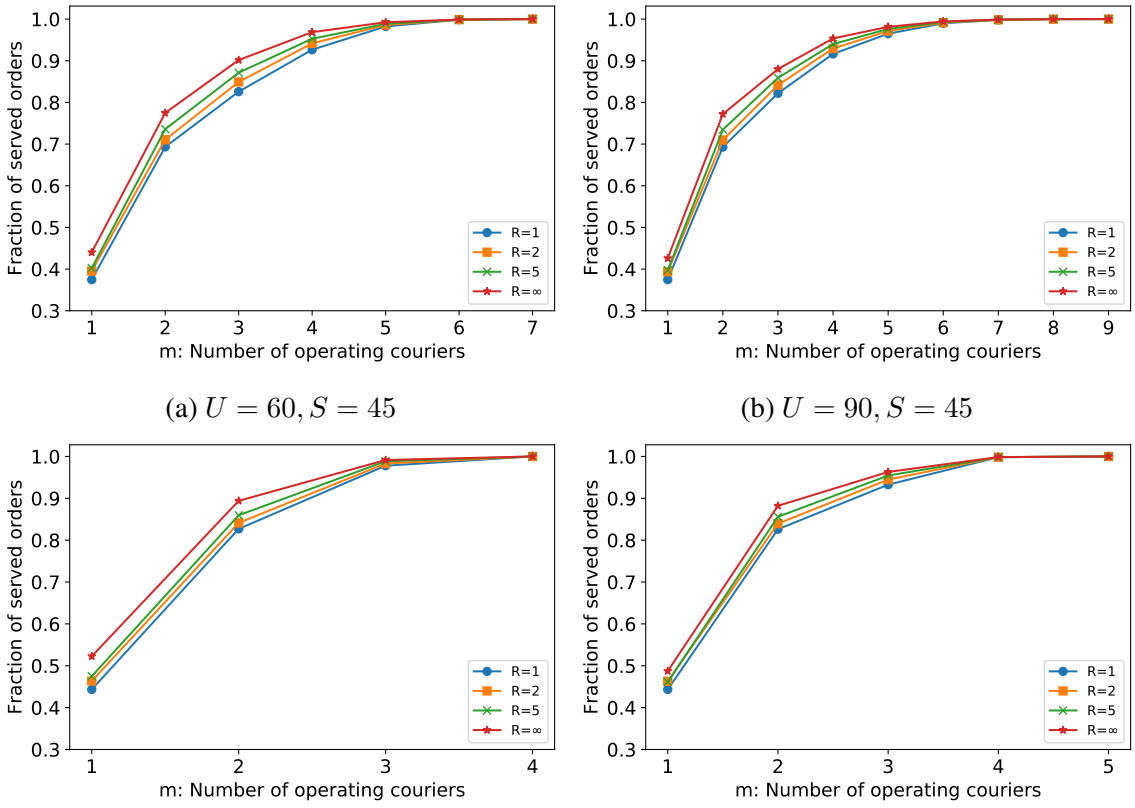
Table 2.10: Minimum fleet size m needed to serve 80% and 95% of all orders on average, for a single depot

	Order coverage		$n = 75$		$n = 150$	
			$S = 45$	$S = 60$	$S = 45$	$S = 60$
$\tau_j \sim Unif(1, 45)$	80%	$R = 1$	7	5	9	6
		$R = 2$	6	4	8	6
		$R = 5$	5	4	8	5
		$R = \infty$	5	3	6	4
	95%	$R = 1$	10	7	14	8
		$R = 2$	9	6	13	8
		$R = 5$	9	6	13	8
		$R = \infty$	8	5	11	7
$\tau_j \sim Tria(1, 1, 45)$	80%	$R = 1$	4	3	5	3
		$R = 2$	3	3	4	3
		$R = 5$	3	3	4	3
		$R = \infty$	3	2	3	3
	95%	$R = 1$	5	4	7	5
		$R = 2$	5	4	7	5
		$R = 5$	5	4	7	5
		$R = \infty$	4	3	6	4

formulation has difficulties solving a large number of replications in reasonable times for relatively larger values of n, thus we limit the results in this section to order volumes of $n \in \{75, 150\}$.

For the tested instances with two depots, the results obtained in terms of the maximum fraction of served orders are similar to the ones from the single depot setting. The greatest benefit is obtained for medium-sized fleets of couriers, as shown in Figure 2.11. For almost every tested fleet size m and values of n, S and U, we observe that increasing the number of radii changes R_d from one to five results in narrowing the performance gap to the upper bound by more than 50%; this is illustrated, for example, by the instances with $m \in \{2, 3\}$ when $U = 90$, where flexibility is most limited. For the largest values of m, performance improvement from increasing R_d is no longer possible since almost every order can be served when $R_d = 1$.

For the results on the fleet sizes required to achieve certain minimum order coverage fractions, we observe that in most cases reducing the fleet size is not possible in this scenario. Indeed, for a coverage requirement of 80% of orders, in almost every scenario the number of couriers needed in the base case coincides with the one from the upper bound, as reported in Table 2.11. In this case, the flexibility provided by sharing couriers between depots is enough to allow either 3 or 2 couriers to cover the required fraction of orders for almost any value of R. When the coverage requirement is increased to 95%, however, the system becomes less flexible, and for a few cases it becomes possible to reduce the required fleet size by slightly increasing R, as shown in Table 2.11.



(a) $U = 60, S = 45$

(b) $U = 90, S = 45$

(c) $U = 60, S = 60$

(d) $U = 90, S = 60$

Figure 2.11: Fraction of served orders with $n = 150$ for two depots

Table 2.11: Minimum fleet size m needed to serve 80% and 95% of orders on average, for two depots

	Order coverage		$n = 75$		$n = 150$	
			$S = 45$	$S = 60$	$S = 45$	$S = 60$
$U = 60$	80%	$R = 1$	3	2	3	2
		$R = 2$	3	2	3	2
		$R = 5$	3	2	3	2
		$R = \infty$	2	2	3	2
	95%	$R = 1$	4	3	5	3
		$R = 2$	4	3	5	3
		$R = 5$	4	3	4	3
		$R = \infty$	4	3	4	3
$U = 90$	80%	$R = 1$	3	2	3	3
		$R = 2$	3	2	3	2
		$R = 5$	3	2	3	2
		$R = \infty$	2	2	3	2
	95%	$R = 1$	4	3	5	4
		$R = 2$	4	3	5	4
		$R = 5$	4	3	5	3
		$R = \infty$	4	3	4	3

2.6 Conclusion

In this chapter we have introduced several integer programs built from time-expanded networks to represent different operational situations in food delivery logistics. The results obtained when solving these optimization problems provide valuable insights on the effect of different parameters on operational performance metrics, namely customer distributions, target delivery time, order volume and fleet size. In particular, this research seeks to answer basic questions about how to optimize the delivery resources in response to different demand patterns and service level requirements and to explore the effectiveness of optimizing the coverage of orders around a depot given a limited delivery capacity as a demand management mechanism. The flexibility of these formulations can easily be adapted to study further trade-offs. We show through computational experiments the interactions between the analyzed metrics for cases with a single and two depots.

This chapter presents simplified instances and network construction algorithms by assuming special geometries to avoid the complexity of routing in a general network. Using these simplifications, we are able to measure the benefits of fleet sizing and demand management in meal delivery settings as this allows us to optimally solve the considered problems for a wide variety of instances. Despite this, a natural line of further research is to adapt our framework to general networks that are more representative of urban settings. As this case not only considers dispatch but also routing decisions, we anticipate that the complexity of the corresponding time-expanded network will make computation prohibitively expensive. Under such circumstances, exploring the use of refinement algorithms like column generation and branch-and-price may provide reasonable research directions for the perfect information case, and adaptive approaches using different dispatch technologies when information is partially revealed over time.

Other interesting extensions of our problems include (i) the study of fleet sizing and demand management in settings involving multiple depots with a shared set of couriers, to analyze how these features scale when more complex settings are considered; and (ii) the study of product substitution and the benefit of being able to select the depot which an order should be picked up from, possibly increasing the efficiency of the delivery process.

CHAPTER 3

COURIER SATISFACTION IN RAPID DELIVERY SYSTEMS USING DYNAMIC OPERATING REGIONS

3.1 Introduction

This chapter considers operational decision-making for rapid delivery networks like those for prepared meals and explicitly introduces approaches to improve courier satisfaction. Customer-centric performance metrics and courier-centric performance metrics will be considered simultaneously. Couriers are primarily motivated by the income and profit they can earn within a delivery network [7]. They also prefer to work in a relatively compact geographic region, likely close to their residence in order to minimize time and cost of commuting. Working in a compact area also provides familiarity advantages including spending shorter times finding parking, restaurants, and delivery locations, and recurrently interacting with the same restaurant workers, leading to overall more efficient service times and a subsequent increase in the number of completed deliveries [77].

To simultaneously achieve both high customer service and high courier satisfaction, we propose a structure for delivery networks in larger geographic areas where the service region is partitioned into multiple smaller *courier regions* designed to reduce or eliminate the likelihood of couriers wandering over the full extent of the service area during a shift. In the context of meal delivery, these courier regions can be created by partitioning the set of restaurants and then allocating (hiring) couriers to serve orders from restaurants in a single courier region. Since such a strategy does eliminate some of the risk-pooling uncertainty management benefits of operating a larger pool of couriers over the full service

area, we further propose that these courier regions be *dynamic*, with boundaries that can be temporarily altered on-demand during the operating day. In our proposed approach for meal delivery, a region is dynamically expanded by adding a number of restaurants from a neighboring region, allowing orders from those origins to be fulfilled by couriers from either the newly-expanded region or from the original region. Limiting the dynamic region changes can hopefully keep most of the courier benefits of small regions while improving courier utilization and customer service performance metrics.

Decisions related to dynamic region reshaping and the assignment of orders to couriers are proposed to be made by a rolling horizon algorithm. The algorithm utilizes ideas from bipartite matching to both construct order-courier assignment recommendations over time while also simultaneously changing some of the restaurant-region assignments (and thus the courier region boundaries).

3.1.1 Main contributions

The main contributions of this research can be summarized as follows:

- We are the first to develop rapid delivery system operational models that explicitly consider courier satisfaction alongside customer satisfaction metrics and studies the tradeoffs between these two types of metrics.

- We propose a dynamic courier region operating strategy for rapid delivery systems, which seeks to capture most of the courier benefits of a static regional partitioning strategy by introducing partial flexibility to balance system objectives when facing uncertainty in order demand.

- We develop a scalable two-phase matching-based rolling horizon algorithm that optimizes key operational decisions for rapid delivery system and successfully integrates dynamic region redefinition and order-courier assignments.

- We demonstrate via an empirical study using real-world data that courier satisfaction metrics can be increased without deteriorating key order service quality metrics for a fixed courier fleet size.

The remainder of the chapter is organized as follows. Section 3.2 provides a brief survey of the related literature. Section 3.3 defines the studied problem, the mathematical notation, the assumptions made and the key performance metrics to measure the quality of solutions. Subsequently, Section 3.4 presents the rolling-horizon matching-based algorithm to solve the problem, while Section 3.5 provides empirical evidence of the effectiveness of the solution approach. Lastly, Section 3.6 summarizes the work done and suggests directions for future research.

3.2 Relevant literature

Management of rapid delivery systems requires solving decision problems such as those modeled by dynamic vehicle routing problems; a comprehensive survey of the extensive literature of this class of decision models and optimization problems can be found in [44, 45]. Furthermore, assignment of couriers to demands in rapid delivery can be classified in the dynamic pickup and delivery problem sub-class (dPDP), which has gained considerable research attention in over the years mostly thanks to the emergence of same-day delivery services, e-commerce, and ride-hailing applications [3, 4, 10, 50].

Like meal delivery, many dPDPs share multiple key characteristics that complicate their solving process, such as limited knowledge of future events (e.g., request placements) and tight service constraints (i.e. urgency of decisions). At the same time, however, these complicating aspects have motivated researchers and practitioners to adopt myopic matching-based rolling horizon algorithms due to its scalability and success at producing high quality solutions, especially in the realm of crowd-sourced transportation applications [6, 23, 27, 47, 70].

Motivated by these technological developments of applications of dPDP, many authors have studied this class of problems both from a theoretical standpoint [5, 8, 48, 62, 74] and in more realistic settings [23, 34, 47, 66]. In the particular case of meal delivery applications, seminal work include the one by [47], which formally introduces the meal delivery routing problem (MDRP), capturing the most crucial aspects of dPDP observed in the meal delivery context. To solve the dynamic version of the problem, the authors propose a framework based on a rolling horizon algorithm that repeatedly solves a matching to make dispatching decisions. The work provides an exhaustive computational study that analyzes the impact of a wide range of operational factors, including the complexity of the dispatching technology, the bundling level, the assignments commitment policy, and specific algorithmic features. In a parallel research effort, [73] attempt to solve the static version of the MDRP to determine the value of having complete information about order arrivals. Due to the extreme complexity of the problem, they develop a framework that simultaneously generates rows and columns to find the optimal solution. By solving a subset of instances from [47], the combined results of both groups of authors determine that in general, a myopic rolling horizon approach is capable of finding near-optimal solutions in terms of service quality.

In a slightly different note, [38] study the operations of a meal delivery system in a setting where couriers are replaced by drones, considering many context-specific constraints that consider features such as order type, drone carrying capacity, and drone battery capacity. The problem is solved using traditional mixed-integer programming techniques embedded in a rolling horizon algorithm. In addition to considering the typical features defining the MDRP, in this chapter we additionally consider defining subregions of the service area to which delivery supply resources (couriers) are allocated to in advance and the management of these courier regions over time. A poor implementation of such an approach could have adverse effects on system performance metrics, since the risk pooling benefits of flexible supply resources may disappear leading to high expected costs [29]. Managing courier

regions over time with dynamic adjustments, as we propose, is in itself a complex stochastic resource allocation problem. Although many dynamic adjustment schemes are possible, the research in this chapter considers a relatively simple scheme in which couriers are allocated *a priori* to regions, but these base courier regions may be expanded temporarily during the operating day to include additional demand (in meal delivery, originating from restaurants). Demand in expanded regions can be served by couriers either from the original base region or from the (now overlapping) expanded region. Given this structure, determining how and when to expand regions or revert them to their base configuration is another problem of stochastic optimization.

The dynamic courier region expansion scheme proposed is an approach to introduce *partial flexibility* to the dynamic delivery resource allocation problem. Although partial flexibility in resource allocation has not been previously studied in the context of crowdsourcing services, it has been widely explored in other areas such as manufacturing. [31] introduce the concept of "*chaining*" in the context of flexible manufacturing and empirically show, through a simulation analysis, that slight improvements in process flexibility (i.e., establishing a *long chain* between resources and jobs) are sufficient to obtain performance comparable to the ideal fully flexible process. In a subsequent paper, [53] characterize the performance of long chains by first proving the supermodularity of the marginal benefits of additional flexible arcs until the long chain is formed, and then using this property to show that the long chain maximizes the system performance among all the possible 2-flexible designs.

3.3 Problem definition

This section presents a model for optimizing rapid pickup-and-delivery operations where orders become known only shortly before they are ready for dispatch and need to be delivered to customers quickly by a target time; meal delivery is the canonical application

example. The model assumes that supply resources (couriers) are operating within the service area, and that they may be restricted to serve orders only within specific regions within the service area at certain times. The model captures the most essential aspects of real-world rapid delivery processes, namely, (i) dynamic order arrivals: *orders* are continuously placed to the system, with no advance information and ready times only minutes after placement times; (ii) multiple pickup locations: many origin locations (*restaurants*) exist, and each order must be delivered from a specific origin; and (iii) dynamic delivery capacity: to complete the delivery of orders, the system employs *couriers* and each is available during some scheduled period of time during the operating day known as their *block*.

Suppose that each courier is assigned *a priori* to a subregion within the service area, which we refer to as their *base courier region*). The courier begins their block in their base region and would prefer to operate within it for most of their block. Base courier regions are defined by partitioning the pickup locations (restaurants), assigning each to a single region in such a way to form contiguous and compact geographical regions. A courier operating in region i can only serve demand originating in region i. While in principle, this definition is most appropriate for systems where delivery locations are relatively close to pickup locations (like meal delivery), it is possible to generalize these ideas.

Such a system configuration prevents couriers from roaming too far away from their base courier region and also incentivizes them to return to that region when idle to maximize the likelihood of matching with a future order; these features can both improve courier satisfaction but also balance resource availability geographically over time. However, it is also possible that service quality may degrade when couriers are paritioned into regions due to a loss of flexibility in order-courier assignments. Therefore, we also allow the decision maker to dynamically and temporarily expand courier regions when doing so will better balance supply and demand. So-called *expanded courier regions* always include the base region pickup locations and then some additional pickup locations in neighboring base

regions. Using an expanded region provides the flexibility to share courier resources, and sharing may improve customer service metrics (*i.e.*, reducing delivery times) at the cost of degrading some courier satisfaction metrics (by increasing travel outside of the base region).

In this study, we model dynamic region reshaping using a discrete approach where each base region can be expanded into one or more neighboring regions by adding pre-defined sets of restaurants in the boundary area and may be contracted by removing these expansions. Thus, at various decision epochs, the decision maker needs to choose which regions to operate in their base configurations and which to operate in various expanded configurations.

3.3.1 Notation and main assumptions

Formally, let D be the set of restaurants comprising a service area, with each restaurant $d \in D$ having an associated two-dimensional location ℓ_d; we use d as the element identifier here since pickup locations are often referred to in the literature as *depots*. Let O be a set of delivery orders, each order $o \in O$ associated with a restaurant d_o, a placement time a_o, a ready time e_o, a two-dimensional delivery location ℓ_o, and a target delivery time s_o. Delivery requests are completed by a set of couriers C, where each courier $c \in C$ is characterized by a start location ℓ_c, a block start time t_c^{start}, and a total block duration Γ_c. Assume that all information about the couriers in C is known in advance. On the other hand, suppose that order information is not known until placement times. Thus, information about the existence of order $o \in O$ becomes available only at time a_o, along with all of the detailed information about the order including its ready time e_o.

Suppose that the system operating day is defined by the time period $\boldsymbol{T} \doteq [0, T]$, $T > 0$. Given $t \in \boldsymbol{T}$, let $O_t \subseteq O$ be the set of active orders at t, *i.e.*, placed orders that have not been delivered by time t, and let $U_t \subseteq O_t$ be the set of active orders not assigned to any

courier by time t. Let $C_t \subseteq C$ be the set of active couriers at time t, *i.e.*, couriers c such that $t \in [t_c^{start}, t_c^{start} + \Gamma_c]$.

The set of base courier regions is denoted by $R \doteq \{r_1, \ldots, r_p\}$. Each base region $r_i \in R$ is defined by a set of restaurants $D_i \subseteq D$; the sets $\{D_i\}_{i=1}^p$ form a partition of D. Furthermore, suppose that expansion sets D_{ij} are also defined in advance. An expansion set D_{ij} contains restaurants $d \in D_j$ that are to be added to base region r_i in the case when that region is expanded in the direction of r_j. We describe a specific approach for defining D_{ij} in Section 3.4.1, but any reasonable approach for defining locations in r_j that are nearby to r_i could be used. Since regions may change over time via expansion actions, let $R'_t \doteq \{r'_{1,t}, \ldots, r'_{p,t}\}$ be the current and possibly expanded courier regions, with $r'_{i,t}$ defined by a set of restaurants $D'_{i,t}$ where for all $t \in \boldsymbol{T}$ and $i \in \{1, \ldots, p\}$, $D_i \subseteq D'_{i,t}$. If $D'_{i,t} \cap D_j \neq \emptyset$ for $i \neq j$, then it is said that $r'_{i,t}$ *is supporting* r_j at time $t \in \boldsymbol{T}$, or equivalently, that $(r'_{i,t}, r_j)$ form a *support pair*. One courier region is supporting another when its assigned couriers may be assigned to orders from a subset of the other's restaurants. We may sometimes also refer to the area of a courier region r_i, which will be defined as the area of contained within the convex hull of the locations of its current restaurants $D'_{i,t}$ at time t.

Courier regions are used to control the operations of couriers. To do this, let $a \geq 0$. We define the *terminal period* of courier c as the last a-duration portion of its block $(t_c^{start} + \Gamma_c - a, t_c^{start} + \Gamma_c] \subseteq [t_c^{start}, t_c^{start} + \Gamma_c]$, and the prior fraction of its block $[t_c^{start}, t_c^{start} + \Gamma_c - a]$ as its *regular period*. Then each courier $c \in C$ operates in region $r'_{\rho_c,t}$ during its regular period $(t_c^{start}, t_c^{start} + \Gamma_c - a]$, available to be assigned to orders originating at restaurants in $D'_{\rho_c,t}$. In contrast, during its terminal period $(t_c^{start} + \Gamma_c - a, t_c^{start} + \Gamma_c]$ each courier is restricted to serve only orders from restaurants in its base (unexpanded) region r_{ρ_c}. When courier c is assigned to an order o, a pickup time is determined which is the later of the courier arrival time to the restaurant and the order ready time e_o. An assignment of c to o

is allowed if the pickup time t falls within the regular period of c and the delivery location ℓ_o lies within the restaurant set $D'_{\rho_c,t}$. If the pickup time is to occur after the regular period but before $t_c^{start} + \Gamma_c$, an assignment is allowed only if $\ell_o \in D_{\rho_c}$, the base region for the courier. Note that the dropoff time for an order is not restricted and may occur after the end of the regular period or the end of the block for any order.

Travel times between locations are encoded in a matrix τ. In addition, a courier that arrives at a restaurant location to pick up an order incurs a service time of $s_p^- + s_p^+$, where s_p^- is the time to park the vehicle, walk to the restaurant and retrieve the order, and s_p^+ the time to leave the restaurant premises walk back to the vehicle and start their delivery route. Likewise, the courier incurs a total service time of $s_d^- + s_d^+$ at a drop-off location when delivering an order, where s_d^- is the time to park the vehicle, walk to the order delivery location and drop the order off, and s_d^+ the time to walk back to their vehicle. We assume that these times are deterministic and invariant over the operating horizon $\boldsymbol{T}$.

To illustrate the pickup and drop-off process, consider a courier $c \in C$ idle at some location ℓ that at time $t \in \boldsymbol{T}$ is assigned to deliver order $o \in O$. The *pickup* of o by c occurs as soon as (i) o is released at the restaurant; and (ii) c arrives at the restaurant location ℓ_{d_o} and enters the restaurant, *i.e.*, exactly at time $t^{pickup} = t + \max\{e_o, \tau_{\ell,\ell_{d_o}} + s_p^-\}$. Once o is picked up, c departs from the restaurant location ℓ_{d_o} at time $t_1^{departure} = t^{pickup} + s_p^+$ to deliver o at drop-off location ℓ_o. The *drop-off* of order o occurs at time $t^{dropoff} = t_1^{departure} + \tau_{\ell_{d_o},\ell_o} + s_d^-$, and c subsequently departs at time $t^{dropoff} + s_d^+$ to continue its operation. After completing the delivery of an order, if courier c has already been assigned to a new order, then they immediately depart toward the restaurant for the new order. Otherwise, courier c repositions by traveling to the restaurant that is the closest (among restaurants in $D'_{\rho_c,t}$ if the dropoff of o occurred during the regular period of c; and among D_{ρ_c} if the dropoff took place during its terminal period) to the location of its last delivery ℓ_o. Nonetheless, at any time during the repositioning, courier c may have its course of action modified if assigned to serve a

new order.

For simplicity, suppose that couriers are compensated using a base pay per time (and thus a fixed compensation given the block duration) as long as they maintain a high order acceptance rate. Under this assumption, we then further assume that couriers accept all assigned orders. Other compensation schemes are possible and could be modeled, for example those where couriers are paid per order delivered and possibly with extra compensation for orders delivered further from their origin restaurants.

In this chapter, we further restrict the problem to one where each courier can deliver only one order at a time from a restaurant. In some scenarios, there may exist operational benefits from consolidating multiple orders into a single delivery route (also known as *order bundling*); clearly, couriers may be able to complete more deliveries per time with bundling, but individual orders may be delivered at larger delays to their placement and ready times. This restriction is made primarily to simplify the analysis; the methodology that we propose for this problem can accommodate order bundling readily if necessary.

3.3.2 Performance metrics

We measure the satisfaction of couriers using the following metrics:

- **First-to-last location travel time (FtL)**: travel time between a courier's start location and its end location.
- **First-to-furthest location travel time (FtMax)**: travel time between a courier's start location and the furthest location contained in its complete route.

Couriers may feel more satisfied if (i) they select the courier region in which they operate; and (ii) they are asked to leave their region infrequently and do not venture far. We decide to capture these ideas using metrics that measure how far couriers venture from their initial locations, and doing so allows comparison to configurations with varying numbers of

courier regions of different sizes.

Improving courier satisfaction, however, may lead to sacrifices in customer service quality since restricting the service area of couriers also reduces the number of feasible order-courier assignments. Hence, we also consider standard customer satisfaction metrics in the existing rapid delivery literature, such as:

- **Percentage of orders delivered**.
- **Click-to-door time (CtD)**: difference between the delivery time of an order and its placement time.
- **Ready-to-door time (RtD)**: difference between the delivery time of an order and its ready time.
- **Ready-to-pickup time (RtP)**: difference between the pickup time of an order and its ready time.

When dynamically reshaping regions, a primary goal is to decrease the workload in courier regions that are overloaded with orders. To measure the workload of a region, we use the **order-per-courier ratio (OPC)**, which we denote as $opc(r)$ for a region r. A specific definition for $opc(r)$ needs to measure the number orders placed over some time period and the number of couriers available during some time period. Using an OPC metric to measure workload has two primary advantages: first, it can be computed very simply and second, it is easy to interpret. The computational study in this chapter will demonstrate that the specific OPC metric that we compute successfully captures the load placed on the available delivery resources in a region and thus guides appropriate load-balancing through the region re-shaping mechanism.

Note that expanding a courier region r_1 results in r_1 supporting some other region r_2, *i.e.*, couriers allocated to r_1 becoming eligible to serve orders from some restaurants in

r_2 in addition to restaurants from r_1. This implies that more orders are to be distributed among couriers from r_1, hence the action of expanding r_1 comes at the expense of increasing $opc(r_1)$ (*i.e.*, increasing its workload); however, the benefit of such action lies in having more resources to serve orders from r_2, in turn decreasing $opc(r_2)$ (*i.e.*, alleviating its workload). On the other hand, if at some point after the expansion of r_1 the performance of r_2 improves enough such that r_2 no longer needs support from r_1, then it might be convenient to shrink back r_1 to prevent its couriers from unnecessarily roaming outside their region of choice. This contraction operation is done based on the resulting area reduction of ongoing supporting regions; for this purpose, we define the **area of a region** r as the area of the convex hull described by the locations of its restaurants.

3.4 A two-stage matching-based rolling horizon algorithm

In this section, we propose an algorithm for assigning couriers to rapid delivery orders and re-shaping courier regions dynamically over time. Other research has found that given the highly dynamic nature of many rapid delivery order processes (such as those for meal delivery) and the urgency of deliveries, operations may not be substantially improved by approaches that attempt to use predictive information about future order arrivals when making order-to-courier assignment decisions [47]. Thus, we propose a rolling horizon algorithm that relies only on known information for order assignments. We furthermore extend the framework of the approach to decide on courier region reshapings over time. At each decision epoch, a set of proposed order-to-courier assignment decisions are determined, but the decisions to be immediately implemented are determined using a so-called *commitment rule*; orders and couriers matched in uncommitted assignments are simply unmatched and reconsidered in the pool by the algorithm again after the horizon rolled for the next decision epoch. Like order-to-courier assignment, we also use ideas from bipartite matching to decide how to shape the courier regions at each decision epoch.

More specifically, the algorithm solves a set of matching optimization problems every f minutes. At decision epoch t:

- A boundary redefinition step first uses a bipartite matching optimization model to decide whether to expand some restaurant sets in $\{D'_{i,t}\}_{i=1}^{p}$ to provide couriers to support other regions with greater workload, maximizing the OPC reduction of supported regions. This step then uses a second bipartite matching to select a subset of ongoing supporting pairs to terminate, shrinking regions in such a way that the area reduction is maximized. This step is described in detail in Section 3.4.1.

- An assignment step solves a third bipartite matching that assigns orders in set U_t to active couriers in set C_t, determining assignment recommendations that minimize freshness loss for orders prior to their pickup. A commitment rule determines which of these recommendations are to be executed. This step is described in Section 3.4.2.

One of the main advantages of using matching models within the algorithm is that finding an optimal weight matching on problems of realistic scale for this application requires little computational effort. This is important in the highly dynamic environments faced by rapid delivery systems, where only a few minutes (at most) may be available to make decisions.

3.4.1 The courier region reshaping step

In this section we describe the expansion and contraction decision process for courier regions. The action of expanding a region is performed to provide support to another region, while the action of contracting a region is taken to terminate unneeded region-to-region supporting pairs. Thus we model the decision of one region providing support to another one as a dichotomy of whether the region to provide the support should start covering a fixed subset of nearby restaurants contained in the region to receive the support, in order to redistribute the excess of workload of the latter. Under this consideration, we model the expansion and contraction operations as a maximum weight bipartite matching between sets

of regions. This simple model is able to simultaneously considers all the possible expansion (contraction) decisions and to select the corresponding actions that maximize the OPC (area) reduction; for this purpose, the weights of matching models provide enough modeling power to capture these criteria. Additionally, the use of matching models presents a practical advantage: modeling an expansion or contraction plan as a region matching limits the number of modifications of each region to no more than one at a time, thus preventing sudden changes in the overall geographical configuration at each iteration of the rolling horizon algorithm.

For the region reshaping step, we consider the following mathematical notation. Let $\epsilon \geq 0$ be a travel time threshold, and for $i \in \{1, \ldots, p\}$, let $\overline{D}_i$ be the geographical center of base region r_i, i.e., $\overline{D}_i = \frac{1}{|D_i|} \sum_{d \in D_i} \ell_d$. For $i, j \in \{1, \ldots, p\}$, the set of restaurants in base region r_j that are covered by $r'_{i,t}$ if $r'_{i,t}$ starts supporting r_j corresponds to $D_{ij} \doteq \{d \in D_j : \tau_{\ell_d, \overline{D}_i} \leq \epsilon\}$. Intuitively, D_{ij} contains all the restaurants of base region r_j whose locations are at an acceptable travel time from base region r_i.

OPC computation. For a courier region r, let $O_t(r)$ be the set of active orders at time $t \in \boldsymbol{T}$ at restaurants contained in r; $U_t(r) \subseteq O_t(r)$ be the set of orders that remain unassigned by time t; and $C_t(r)$ the set of active couriers at time t allocated to region r. Algorithm 2 shows the routine that computes the OPC of a region. For region index i and time t, the routine first verifies that region $r'_{i,t}$ has active couriers in it. If that is not the case, then $opc(r'_{i,t})$ is assigned a very large value, represented by ∞, to give a substantial penalty to regions without active delivery resources (lines 3 and 4). Otherwise, it counts the number of active couriers allocated to base courier region r_i, increasing the count by one for each courier in its regular period, and by a discounted amount for each courier in its terminal period, as such a courier is eligible to serve just a fraction of the orders in $r'_{i,t}$ (lines 5 - 9). The algorithm then counts the number of active orders in $r'_{i,t}$ that are or may be assigned to a courier allocated to r_i: for each unassigned order $o \in U_t(r'_{i,t})$, the count is

increased by a discounted amount based on the number of regions covering restaurant d_o; for each assigned orders, the count is increased by one unit only if the courier assigned to it is allocated to r_i, as otherwise such order does not make use of the region's delivery resources (lines 10 - 14).

Algorithm 2 (OPC_COMPUTATION)

Input: Region index i, time stamp t.
Output: Order-per-courier ratio $opc(r'_{i,t})$.

1: $n_{orders} \leftarrow 0$
2: $n_{couriers} \leftarrow 0$
3: **if** $C_t(r_i) = \emptyset$ **then**
4: **return** ∞
5: **for** $c \in C_t(r_i)$ **do**
6: **if** $t > t_c^{start} + \Gamma_c - a$ and $O_t(r'_{i,t}) \neq \emptyset$ **then**
7: $n_{couriers} \leftarrow n_{couriers} + \frac{|O_t(r_i)|}{|O_t(r'_{i,t})|}$
8: **else**
9: $n_{couriers} \leftarrow n_{couriers} + 1$
10: **for** $o \in O_t(r'_{i,t})$ **do**
11: **if** $o \in U_t(r'_{i,t})$ **then**
12: $n_{orders} \leftarrow n_{orders} + \frac{1}{|\{i \in \{1,\ldots,p\} : d_o \in D'_{i,t}\}|}$
13: **else if** o is already assigned to some $c \in C_t(r_i)$ **then**
14: $n_{orders} \leftarrow n_{orders} + 1$
15: **return** $\frac{n_{orders}}{n_{couriers}}$

Region workload assessment. To evaluate the expansion or contraction of regions, we require a classification criteria of whether a region is eligible to provide or receive support. Although many different rules may be defined for this purpose, we use a simple logic that considers a threshold on the OPC value to perform such region classification. More precisely, let $\overline{OPC} > 0$ be a threshold on the OPC values. Then we partition the set of expanded regions into the subsets $R'^{+}_t \triangleq \{i \in \{1,\ldots,p\} : opc(r'_{i,t}) \leq \overline{OPC}\}$ and $R'^{-}_t \triangleq \{i \in \{1,\ldots,p\} : opc(r'_{i,t}) > \overline{OPC}\}$, which respectively define regions whose reduced workload makes them eligible to provide support to other regions, and regions whose workload levels exceed a maximum acceptable threshold and so require support from other regions.

Region expansion matching model. We model this step as a maximum weight bipartite matching problem with disjoint node sets R_t^+ and R_t^-. Each arc represents the decision of enabling the corresponding support pair, and its weight captures the associated reduction in the OPC of the supported region. The practical motivation of this operation is to distribute the workload between regions more evenly, in particular when two or more neighboring regions have an elevated workload disparity.

Let $t \in \boldsymbol{T}$ be the time at which the expansion step is performed. For each $i \in R_t'^+$, let $R_t'^-(i) \doteq \{j \in R_t'^- : D_{ij} \neq \emptyset\}$ be the set of indices of regions that are eligible to start receiving support from $r'_{i,t}$. For every potential (i.e. not currently active) support pair $(r'_{i,t}, r'_{j,t}), i \in R_t'^+, j \in R_t'^-(i)$, let $\Delta_{i,j}^{opc}$ denote the decrease of $opc(r'_{j,t})$ that would result from enabling support from $r'_{i,t}$ to $r'_{j,t}$. To mathematically formulate the problem, consider the following decision variables:

$$z_{i,j} = \begin{cases} 1 & \text{if } r'_{i,t} \text{ is selected to start supporting } r'_{j,t}, \quad \forall i \in R_t'^+, \ \forall j \in R_t'^-(i) \\ 0 & \text{otherwise} \end{cases}$$

The expansion step solves the following optimization program:

$$\max \sum_{i \in R_t'^+} \sum_{j \in R_t'^-(i)} \Delta_{i,j}^{opc} z_{i,j} \tag{3.1a}$$

$$\text{s.t.} \sum_{j \in R_t'^-(i)} z_{i,j} \leq 1, \qquad \forall i \in R_t'^+ \tag{3.1b}$$

$$\sum_{i : j \in R_t'^-(i)} z_{i,j} \leq 1, \qquad \forall j \in R_t'^- \tag{3.1c}$$

$$z_{i,j} \in \{0,1\}, \qquad \forall i \in R_t'^+, \ \forall j \in R_t'^-(i) \tag{3.1d}$$

Objective (3.1a) seeks to maximize the total OPC reduction of regions that start receiving support. Constraints (3.1b) require that each potential supporter starts providing support to at most one eligible region, and Constraint set (3.1c) states that each new region seeking

support may start receiving assistance from no more than one new eligible supporter; for each region, we limit support changes to at most one at a time to prevent severe changes in the size of regions. For optimal solution vector z^*, any pair of region indices (i, j) such that $z^*_{i,j} = 1$ translates into region $r'_{i,t}$ starting to cover some of the restaurants in r_j, i.e., the restaurant set $D'_{i,t}$ covered by $r'_{i,t}$ incorporating the set of restaurants D_{ij}.

Region contraction matching model. This step also solves a maximum weight bipartite matching with disjoint node sets corresponding to region sets; however, each arc now represents an ongoing support pair, and its weight corresponds to the area reduction from severing the associated ongoing support. From a practical perspective, this procedure allows to maintain courier regions as compact as possible, by contracting regions whose expansion is no longer required to maintain the workload of some neighbors under the considered threshold, thereby preventing couriers from unnecessarily roaming outside their base courier region.

Let $t \in \boldsymbol{T}$ be the time at which the contraction step is performed. For each region $i \in R'_t$, let $S^+_t(i) \subseteq R'^+_t$ be the set of indices of regions $r'_{j,t}$ receiving support from $r'_{i,t}$ by time t, such that if the support from $r'_{i,t}$ to $r'_{j,t}$ is interrupted, the resulting value of $opc(r'_{j,t})$ would remain below the threshold $\overline{OPC}$. For each pair $(i, j), i \in R'_t, j \in S^+_t(i)$ let $\Delta^{area}_{i,j}$ be the area reduction of region $r'_{i,t}$ that would result from terminating the support provided to $r'_{j,t}$. The mathematical formulation of the problem considers the following decision variables:

$$y_{i,j} = \begin{cases} 1 & \text{if the support from } r'_{i,t} \text{ to } r'_{j,t} \text{ is interrupted,} \quad \forall i \in R'_t, \ \ \forall j \in S^+_t(i) \\ 0 & \text{otherwise} \end{cases}$$

The contraction step then solves the integer program given by

$$\max \sum_{i \in R'_t} \sum_{j \in S_t^+(i)} \Delta_{i,j}^{area} y_{i,j} \tag{3.2a}$$

$$\text{s.t.} \sum_{j \in S_t^+(i)} y_{i,j} \leq 1, \qquad \forall i \in R'_t \tag{3.2b}$$

$$\sum_{i \in R'_t : j \in S_t^+(i)} y_{i,j} \leq 1, \qquad \forall j \in R'^{+}_t \tag{3.2c}$$

$$y_{i,j} \in \{0,1\}, \qquad \forall i \in R'_t, \ \ \forall j \in S_t^+(i) \tag{3.2d}$$

Objective (3.2a) maximizes the total area reduction from severed supports. As in the expansion model, Constraint sets (3.2b) and (3.2c) restrict support changes in each region to at most one, to prevent drastic changes in the current region sizes.

Given optimal solution vector y^*, pairs of region indices (i, j) such that $y^*_{i,j} = 1$ translates into region $r'_{i,t}$ terminating the support provided to r_j, i.e., the set of restaurants D_{ij} being removed from the restaurant set $D'_{i,t}$ covered by $r'_{i,t}$.

3.4.2 The order-courier assignment step

Given the courier region configuration resulting from the region reshaping step, this second step constructs next pickup recommendations for each courier by solving a maximum weight bipartite matching problem; the optimization problem is formulated as a bipartite graph whose node sets correspond to the set of unassigned active orders U_t and the set of operating couriers C_t, and where each arc represents a feasible assignment between an order and a courier. Each potential assignment has an associated weight that captures the loss of freshness of the associated order before the courier picks it up.

The matching model. Let $t \in \boldsymbol{T}$ be the assignment step execution time. For each order $o \in U_t$, let $C_t(o)$ be the set of couriers that at time t are eligible to be assigned to deliver

o, i.e., couriers allocated to some region $r'_{i,t}$ containing restaurant d_o that are able to pick up order o before their blocks end; and further consider the set of eligible order-courier assignments $\mathcal{A}_t \doteq \{(o,c) : o \in U_t, c \in C_t(o)\}$. This step determines whether or not each unassigned order $o \in O_t^u$ should be assigned to some courier in $C_{o,t}$, and if so which courier it is assigned to. Thus, we make use of the following decision variables:

$$x_{o,c} = \begin{cases} 1 & \text{if order } o \text{ is assigned to courier } c \text{ for delivery,} \quad \forall (o,c) \in \mathcal{A}_t \\ 0 & \text{otherwise} \end{cases}$$

$$w_o = \begin{cases} 1 & \text{if order } o \text{ remains unassigned after the current execution,} \quad \forall o \in U_t \\ 0 & \text{otherwise} \end{cases}$$

For each feasible assignment $(o,c) \in \mathcal{A}_t$, let $t_{o,c}^{pickup}$ be the pickup time of order o if assigned to courier c. The weight associated with assignment (o,c) is then computed as $t_{o,c}^{pickup} - e_o$. This term corresponds to the RtP and measures the freshness loss incurred by order o while awaiting pickup if assigned to courier c, due to a potential mismatch between its ready time and its pickup time. Consequently, we determine the assignment recommendations by solving the following assignment problem:

$$\min \quad \sum_{o \in U_t} \bar{b}_o w_o + \sum_{(o,c) \in \mathcal{A}_t} (t_{o,c}^{pickup} - e_o) x_{o,c} \tag{3.3a}$$

$$\text{s.t.} \quad w_o + \sum_{c \in C_t(o)} x_{o,c} = 1, \qquad \forall o \in U_t \tag{3.3b}$$

$$\sum_{o \in U_t : c \in C_t(o)} x_{o,c} \leq 1, \qquad \forall c \in C_t \tag{3.3c}$$

$$w_o, x_{o,c} \in \{0,1\}, \qquad \forall o \in U_t, \;\; \forall c \in C_t(o) \tag{3.3d}$$

Objective (3.3a) minimizes the total freshness loss of the selected assignments prior to the pickup of the associated orders; additionally, a weight $\bar{b}_o$ is associated with the decision

of leaving order o unassigned, which satisfies $\bar{b}_o > \max_{c \in C_t(o)} t^{pickup}_{o,c} - e_o$ to encourage assigning as many orders as possible. Constraints (3.3b) require that every unassigned order is either assigned to an eligible courier or left unassigned until the next iteration of the rolling horizon algorithm. In turn, Constraint set (3.3c) ensures that every courier receives at most one assignment recommendation.

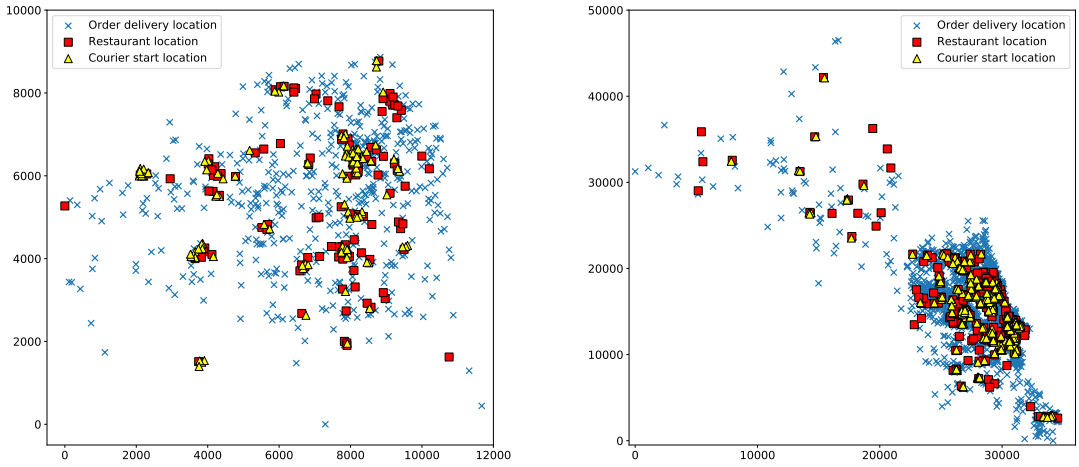
The recommendation confirmation criteria. To reduce the effect of uncertainty in the outcome of the assignment decisions, we employ a simple confirmation criteria that avoids executing the least urgent assignment recommendations. In particular, given an optimal assignment vector x^*, for each feasible assignment (o, c) such that $x^*_{o,c} = 1$, the instruction of serving order o is communicated to courier c only if o becomes ready for pickup and c finishes its last scheduled assignment before the next optimization, i.e., if $e_o < t + f$ and $e_c < t + f$.

3.5 Experimental results

In this section we present the performance results obtained from applying the proposed framework to two instances from the Grubhub MDRP instance repository (https://github.com/grubhub/mdrplib), devised using real-world daily historic data from different metropolitan areas. In accordance to the nomenclature used by the repository, the particular instances we consider correspond to:

- *0o100t100s2p100*: a relatively small instance with 505 orders, 116 restaurants, and 117 couriers. We refer to this instance to as *Instance A*.
- *9o100t100s2p100*: a relatively large instance with 1746 orders, 270 restaurants, and 432 couriers. We refer to this instance to as *Instance B*.

Figure 3.1 shows the restaurant locations, the order delivery locations, and start locations of couriers for each considered instance.



(a) Instance A (b) Instance B

Figure 3.1: Visualization of instance setups

In all the instances, we assume times are non-negative integers and measured in minutes. The operating horizon $\boldsymbol{T}$ represents a day-long operating period of 900 minutes. For two locations $\ell = (x_1, y_1)$ and $\ell' = (x_2, y_2)$, we compute the travel time from ℓ to ℓ' as $\tau_{\ell,\ell'} = \left\lceil \frac{1}{\nu}\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \right\rceil$, where ν is the couriers' speed, set to $\nu = 320$ meters per minute. We consider pickup and drop-off service times to be $s_p^- = s_p^+ = s_d^- = s_d^+ = 2$ minutes, and set the target delivery time of each order $o \in O$ to $s_o = a_o + 40$ minutes. The rolling horizon algorithm is executed every $f = 5$ minutes.

Each instance is run employing multiple numbers of regions p. Parameter values for $\epsilon, \overline{OPC}$ and courier terminal period duration a are also instance-dependent and we select them via parameter tuning. Note that the parameter ϵ has practical meaning only if $p \geq 2$, while the parameters $\overline{OPC}$ and a are relevant only for settings with dynamic courier regions. In general, the conducted experiments include running the proposed algorithms for the following configurations:

- $(p = 1)$ (*i.e.*, a single region);

- $(p \geq 2, \epsilon = 0)$ (*i.e.*, the multiple static regions); and
- $(p \geq 2, \epsilon > 0)$ (*i.e.*, the multiple regions with dynamic boundaries)

The configuration $(p = 1)$ corresponds to the case where every order can be assigned to any on-duty courier, and hence its also referred to as the fully flexible case; as such, it is expected that this scenario attains the best diner satisfaction (*i.e.*, lowest CtD, RtD and RtP), although possibly having couriers roaming across the entire region, incurring an overall deterioration of the satisfaction of couriers. On the contrary, the scenario $(p \geq 2, \epsilon = 0)$ represents the most restrictive case in terms of order-courier assignments since couriers are limited to only one of the p static base courier regions, and therefore it is expected to observe a service quality degradation. However, the fact that couriers operate in smaller localized regions should produce overall greater levels of courier satisfaction (*i.e.*, lower FtL and FtMax). The last scenario $(p \geq 2, \epsilon > 0)$ illustrates a hybrid approach that comprises a set of p base courier regions to produce a better courier satisfaction, whose boundaries may be temporarily enlarged in an on-demand basis to improve the overall system delivery performance. By applying this hybrid approach to Instance A and Instance B, we seeks to empirically show that employing dynamic courier regions to add a limited amount of flexibility to the system suffices to achieve high satisfaction levels for both couriers and diners, and without having to add extra couriers.

When reporting the results of a specific configuration for which the system is unable to deliver all the orders by the end of the operating horizon $\boldsymbol{T}$, we replace the CtD, RtD, and RtP values of each undelivered order by the corresponding maximum value among the orders successfully delivered in the same run, this in order to perform comparisons between configurations using the same set of orders.

The base courier region construction algorithm employed when $p \geq 2$ may be found in Appendix B.1. Once the regions are constructed, each of the couriers in each instance is

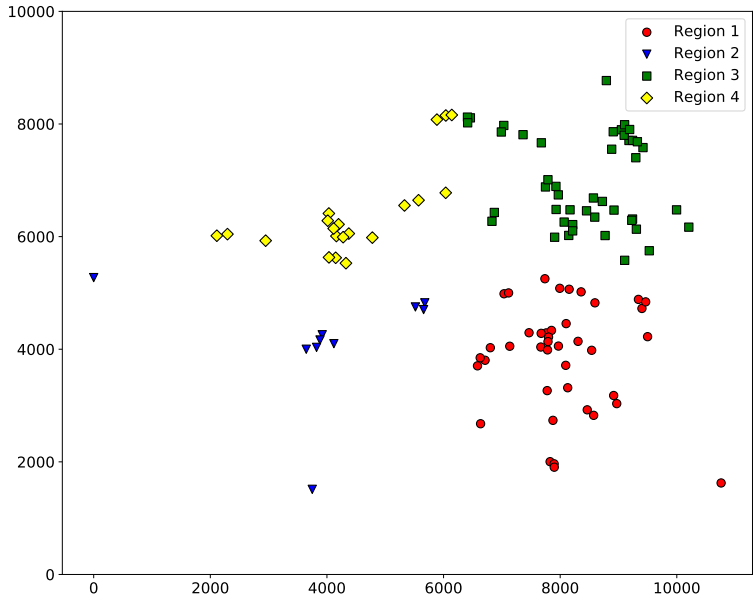


Figure 3.2: Partition of restaurants from Instance A into $p = 4$ base courier regions

Table 3.1: Parameter values used for simulating different number of dynamic regions for Instance A

p	2	3	4	5
ϵ	25	25	25	40
$\overline{OPC}$	1.8	1.8	1.8	1.8
a	10	10	10	20

allocated to the region that contains the restaurant that is closest to its start location.

3.5.1 Instance A

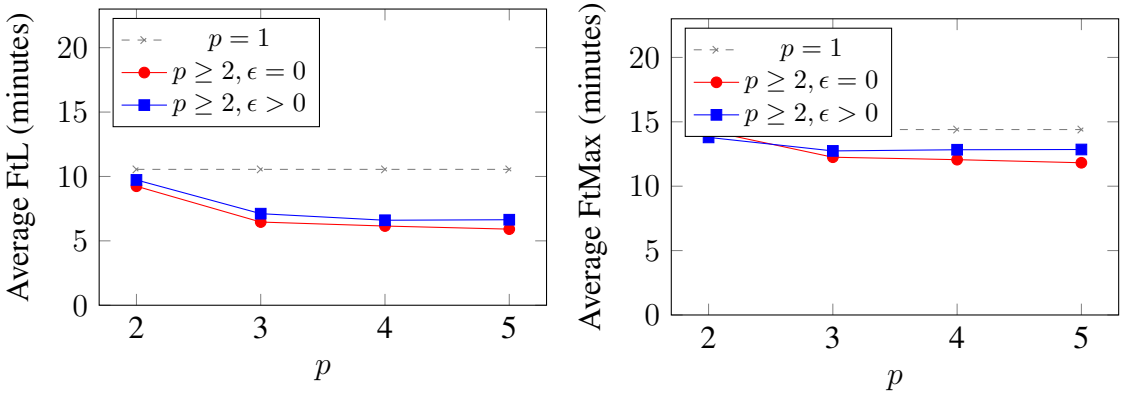
Figure 3.1a illustrates the setup of Instance A, and Figure 3.2 provides the resulting base courier regions for a partition with $p = 4$ regions. Overall, the restaurant and order delivery locations are homogeneously distributed over the considered territory. For the scenario $(p \geq 2, \epsilon > 0)$, Table 3.1 shows the parameter configuration employed for each value of p. We obtain them by performing a tuning process of the parameters for each value of p, using a grid search over the ranges of values in Table 3.2 and selecting the combination that yields the most balanced performance in terms of CtD and FtL among all the non-dominated pairs of values.

Table 3.2: Sets of values employed in the parameter tuning for Instance A

ϵ	$\overline{OPC}$	a
$\{5, 10, 15, 20, 25, 30, 40\}$	$\{1.0, 1.2, 1.5, 1.8, 2.0\}$	$\{0, 10, 20, 30\}$

Figure 3.3 shows the obtained courier-related metric values for the different considered scenarios. Naturally, increasing the number of regions p results in smaller base courier regions, which contributes to producing courier routes that are significantly more localized around their start locations. This is reflected in the reductions of the average FtL and FtMax with respect to the most flexible scenario ($p = 1$), which respectively ascend to approximately 40% and 12% for both scenarios ($p \geq 3, \epsilon = 0$) and ($p \geq 3, \epsilon > 0$), see Figures 3.3a and 3.3b. However, more compact courier regions also produces a loss in flexibility when assigning orders to couriers, ultimately causing a negative impact in the system delivery performance. This is illustrated in Figure 3.4, which shows the average values of order-related metrics obtained for the different configurations. The findings show that when $\epsilon = 0$ and p is large enough, the system is no longer able to serve all the orders, see Figure 3.4a; this is explained by the loss of flexibility from imposing static region boundaries in the set of feasible order-courier assignments, which is more pronounced for greater values of p. Even for the cases where the system successfully serves all the orders, the rigidity of setting $\epsilon = 0$ deteriorates all the delivery time metrics; in particular, the average CtD experiences a deterioration of 17.7% with respect to the fully flexible scenario ($p = 1$). Nonetheless, the slightly more flexibility from considering dynamic courier regions considerably reduces the service quality loss: for all the tested values of p, incorporating the region reshaping step in the simulation results in service of the complete set of orders, with an average CtD increment of up to 1.6% with respect to ($p = 1$), just a negligible fraction of the quality loss experienced with $\epsilon = 0$.

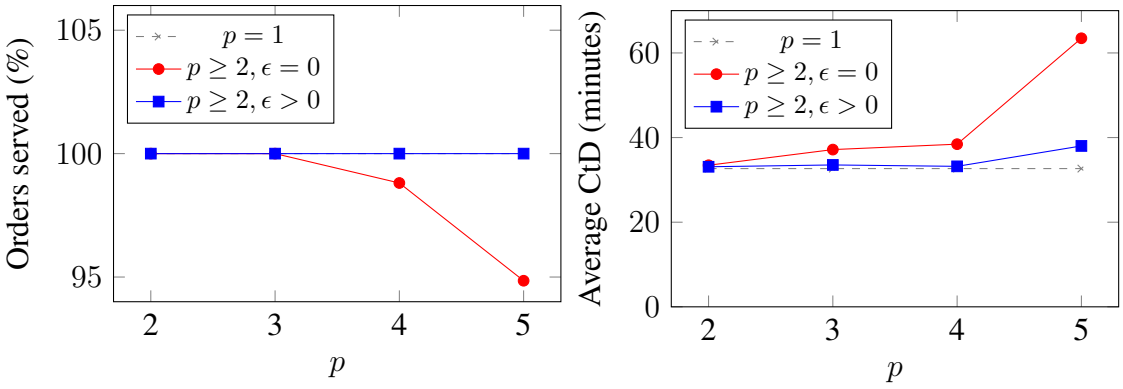
These positive results are shown in detail in Figure 3.5, which confirms that dynamic courier regions achieve very similar CtD levels to the fully flexible case ($p = 1$), while pro-



(a) Average FtL

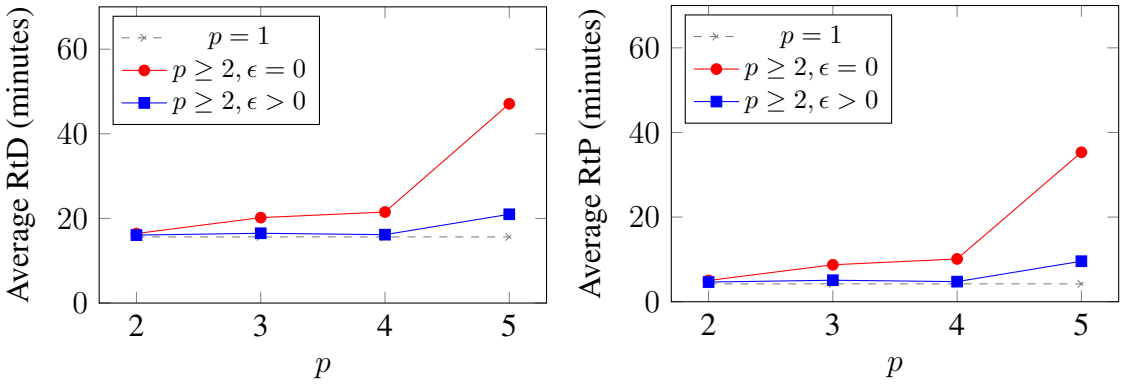
(b) Average FtMax

Figure 3.3: Comparison of courier statistics between different configurations for Instance A



(a) Percentage of orders delivered

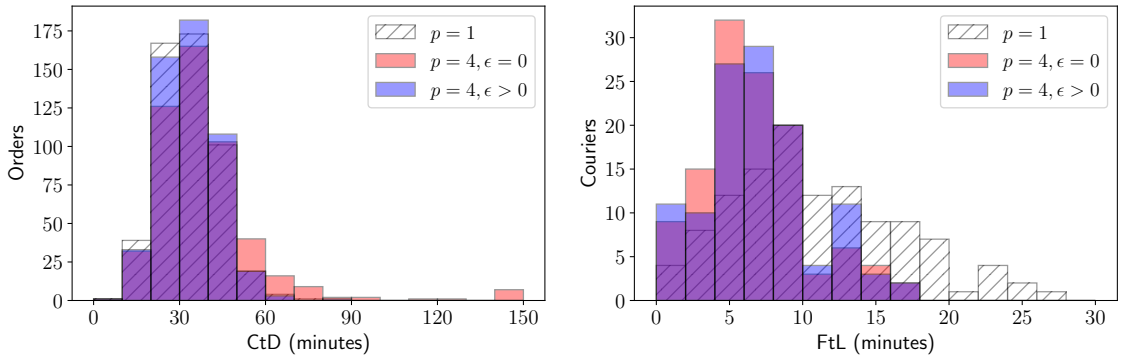
(b) Average CtD



(c) Average RtD

(d) Average RtP

Figure 3.4: Comparison of order statistics between different configurations for Instance A



(a) CtD distribution

(b) FtL distribution

Figure 3.5: Distributions of order and courier satisfaction metrics for Instance A considering $p \in \{1, 4\}$ regions

ducing routes almost that are almost as compact as the most restrictive case $(p = 4, \epsilon = 0)$. More specifically, the median FtL of $(p = 4, \epsilon = 0)$ and $(p = 4, \epsilon > 0)$ are almost identical and also 33% less than the median FtL of $(p = 1)$. On the other hand, although the median CtD is similar in all scenarios, the 90-percentile of CtD in the cases $(p = 1)$ and $(p = 4, \epsilon > 0)$ are equivalent and 14.5% smaller than the most restrictive case $(p = 4, \epsilon = 0)$.

3.5.2 Instance B

The setup of Instance B is shown in Figure 3.1b, and a 9-region partition of this instance is provided as an example in Figure 3.6. This instance comprises a considerably larger order volume than Instance A, and unlike the latter, restaurants are no longer homogeneously distributed across the considered territory. The parameters used for the case $(p \geq 2, \epsilon > 0)$ are listed in Table 3.3, which we obtain via parameter tuning similarly as for Instance A, using the value sets in Table 3.4.

Despite the differences in geography and order volume with respect to Instance A, the findings for Instance B suggest that the benefits are similar in terms of both courier satisfaction and service quality when employing dynamic courier regions. The courier satisfaction met-

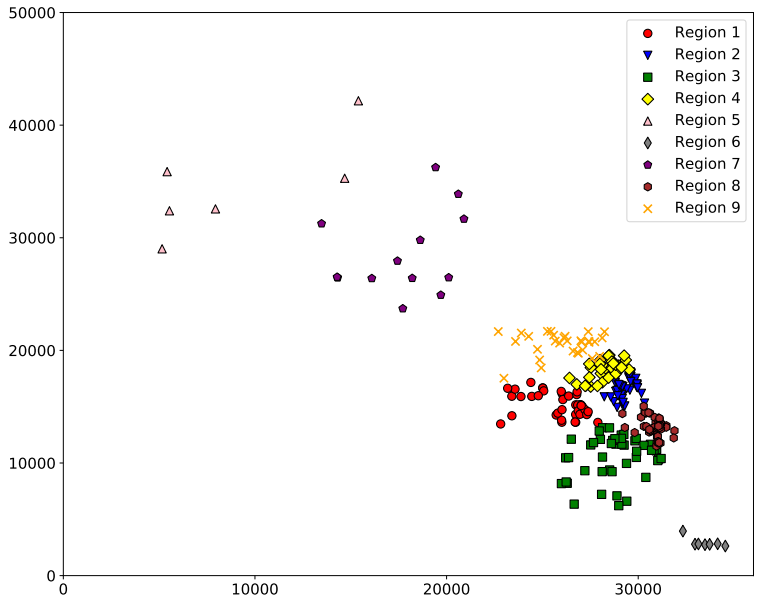


Figure 3.6: Partition of restaurants from Instance B into $p = 9$ base courier regions

Table 3.3: Parameter values used for simulating different number of dynamic regions for Instance B

p	2	3	4	5	6	7	8	9	10	11	12
ϵ	60	70	70	70	60	50	50	50	50	50	60
$\overline{OPC}$	1.0	1.2	1.5	1.2	1.2	1.5	1.8	1.8	1.8	1.8	1.8
a	20	0	30	0	20	10	0	10	0	0	20

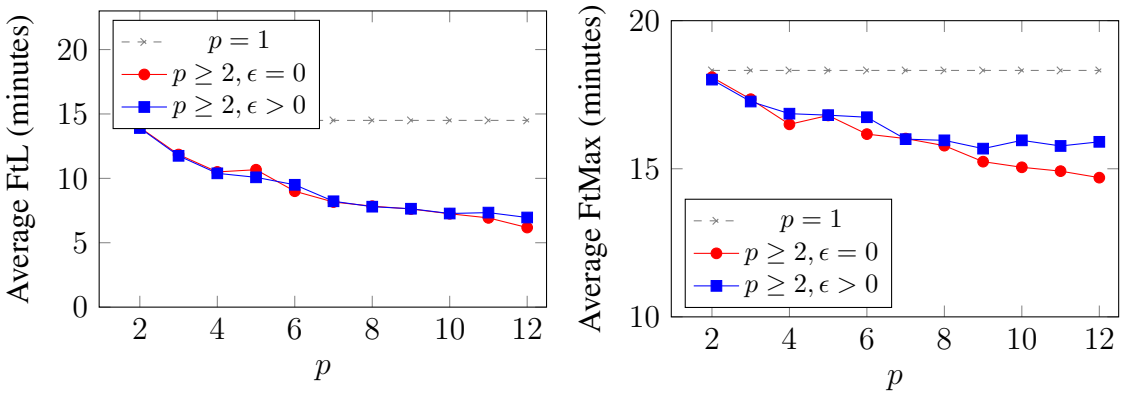
Table 3.4: Sets of values employed in the parameter tuning for Instance B

ϵ	$\overline{OPC}$	a
$\{10, 20, 30, 40, 50, 60, 70, 80\}$	$\{1.0, 1.2, 1.5, 1.8, 2.0\}$	$\{0, 10, 20, 30\}$

rics obtained from solving this instance are shown in Figure 3.7 for different partition sizes p. As an increase of p reduces the size of courier regions, this also decreases both the average FtL and FtMax with respect to the case ($p = 1$) by enforcing couriers to serve only orders from their base region, unless preserving service quality dictates otherwise if $\epsilon > 0$. Interestingly, the benefit is in general almost identical for both $\epsilon = 0$ and $\epsilon > 0$, achieving an average FtL reduction of more than 50%, and a decrease of 15% of the average FtMax with respect to ($p = 1$).

Note that for $p \geq 9$, the benefit in FtMax when $\epsilon > 0$ is slightly less than $\epsilon = 0$; however, having $\epsilon > 0$ preserves the system's ability to serve every placed order, which is not the case for $p \geq 9, \epsilon = 0$, see Figure 3.8a: the loss of flexibility in order-courier assignments when p is large enough and $\epsilon = 0$ makes it impossible to serve every order with the resulting courier initial allocation. However, inserting partial flexibility by increasing ϵ (*i.e.*, by allowing dynamic redefinition of region boundaries) restores such capability; the price to pay comprises an increase in the average CtD of less than 6%, which is more than acceptable considering the associated reductions in average FtL and FtMax. Furthermore, the findings are also favorable in terms of order-related metrics for $p \leq 8$, as shown in Figure 3.8: when considering multiple base courier regions, the CtD deterioration with respect to the fully flexible case $(p = 1)$ can be decreased by between 50% and 90% when switching from static to dynamic courier regions, ultimately achieving an average CtD increase of as little as 1.5% when $\epsilon > 0$.

A more detailed view of the results in terms of CtD and FtL is provided in Figure 3.9 for $p \in \{1, 9\}$. In general, the CtD distribution is similar for all three cases, except for the fact that $(p = 9, \epsilon = 0)$ is not able to serve all the orders. Moreover, there exist differences between the three scenarios in the upper 1.5% of observed CtD: for these orders, a setup with static regions achieves a CtD that is 32% larger than the associated CtD for $p = 1$, whereas having regions with dynamic boundaries reduces that gap to a mere 6%. On the contrary, there are



(a) Average FtL

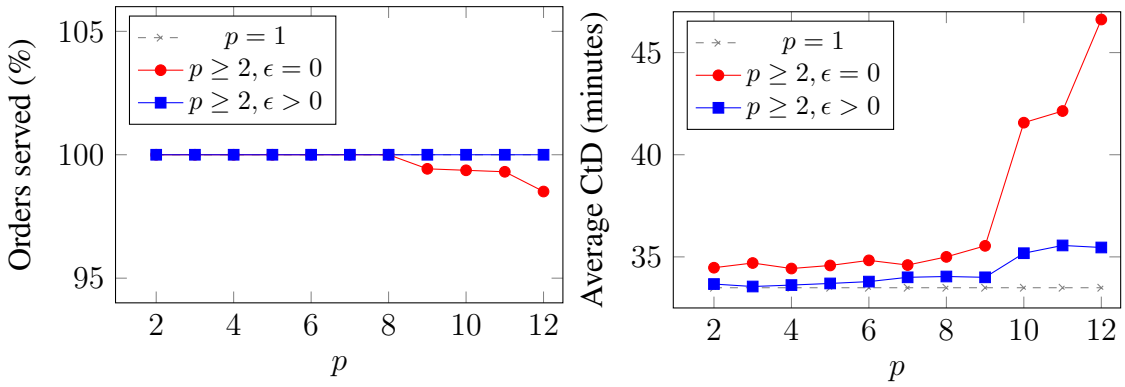
(b) Average FtMax

Figure 3.7: Comparison of courier statistics between different configurations for Instance A

more pronounced differences in the observed FtL distribution, specifically between $p = 1$ and $p = 9$. In particular, having regions with dynamic boundaries achieves a median FtL that is 54% lower than when $(p = 1)$, and surprisingly, 14% lower than $(p = 9, \epsilon = 0)$. This last outcome might occur when the start location of couriers of an expanded region are closer to the delivery locations of some order from the region receiving support than the latter's own couriers. As a result, employing the couriers from the expanded region might result in better courier satisfaction than assigning these orders to courier in the supported region. Lastly, the 95-percentiles of FtL for the configurations with both static and dynamic multiple courier regions are identical and 47% lower than the value observed for $p = 1$.

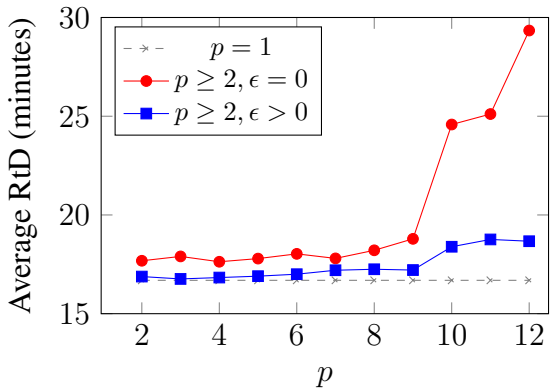
3.5.3 Discussion

By applying the hybrid approach to both Instance A and Instance B, we show that the additional flexibility of dynamic courier regions makes possible to attain diner satisfaction levels that are very similar to the fully flexible case of a large single region, while achieving courier satisfaction levels comparable to the most restricted case of multiple smaller static regions. This provides significant practical value as it implies that when posting blocks

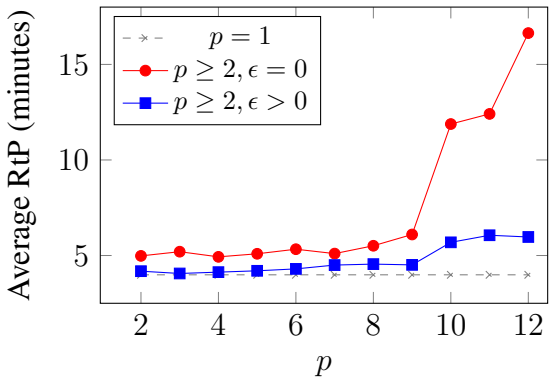


(a) Percentage of orders delivered

(b) Average CtD

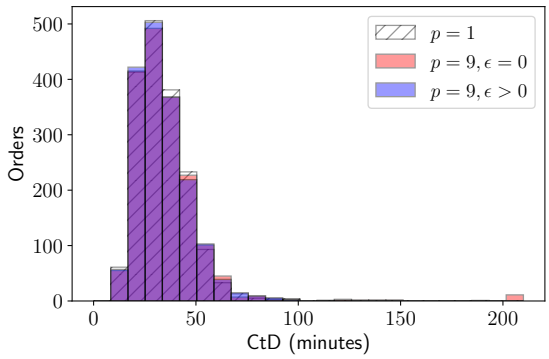


(c) Average RtD

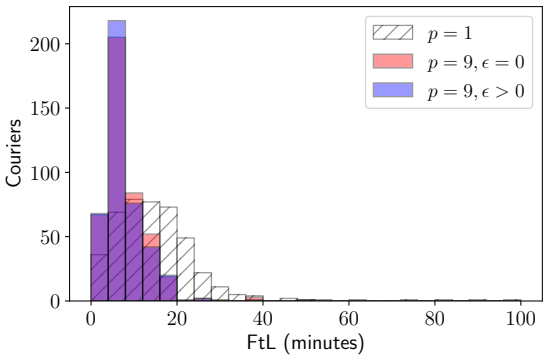


(d) Average RtP

Figure 3.8: Comparison of order statistics between different configurations for Instance B



(a) CtD distribution



(b) FtL distribution

Figure 3.9: Distributions of order and courier satisfaction metrics for $p \in \{1, 9\}$ regions

to be booked by couriers, the planner may be able specify not only the block start time and duration but also the area (*i.e.*, the set of restaurants) where the courier would operate during most of (if not all) its block, without this implying a significant deterioration of diner satisfaction.

Another aspect that we believe is important to consider is that when operating multiple small regions, static region boundaries offers a better control of the areas where each courier operates, in turn increasing the overall courier satisfaction. Nonetheless, this advantage comes at a greater risk of deteriorating the quality of service since under this setting, the planner must determine an effective allocation of courier blocks for all the regions. This requires estimating both order volume and placement times for each individual region, making the overall courier planning process more susceptible to unexpected changes in demand: failure to predict sudden changes in the order volume or placement time in one or more regions may translate in substantially more costs due to undelivered orders and/or lower courier utilization. Such risk, however, is greatly reduced when employing dynamic courier regions, as sharing delivery resources between regions may dynamically correct imbalances in their workloads that may originate from miscalculations in the courier planning process. This is in fact what we observe in the experiments: dynamic courier regions allow to serve all the orders for values of p for which static courier regions fail to do so, with little decrease in service quality when compared to the fully flexible case of operating a single large region.

3.6 Conclusion

This chapter constitutes the first attempt at incorporating the satisfaction of couriers in the context of meal delivery operations, under the assumption that couriers in general prefer to operate in localized and defined areas. Furthermore, we study the trade-off between courier satisfaction and the most traditional diner-centric satisfaction metrics found in the

existing literature. We achieve this by introducing the notion of dynamic courier regions: small operating regions to which couriers are allocated to and whose boundaries may be enlarged in an on-demand fashion. The small sizes of these regions limit the operation of couriers to localized areas, thus improving their satisfaction; whereas the dynamic nature of courier regions seeks to reduce the negative effect of this restricted allocation in the overall service quality. The inherent dynamism and urgency of meal delivery operations allows us to solve the problem using a rolling horizon approach that repeatedly solves a sequence of bipartite matching problems to determine both the region boundary redefinition and the order-courier assignment decisions.

By applying this framework to real-world instances of different sizes and geographical distributions, we demonstrate the effectiveness of the proposed approach: the adoption of multiple small regions with dynamic boundaries excels at producing overall well-balanced solutions, with service quality levels on-par with operating a single large region, and courier satisfaction levels akin to the most restricted case with small invariant regions. Moreover, the capability of dynamically reshaping regions successfully mitigates the risk of degrading service quality due to errors in the courier planning process compared to operating multiple static regions.

We believe that there are multiple research directions that are worth exploring. An intuitive next step is to evaluate the value of considering a set of "unrestricted" couriers, namely a fraction of couriers with the ability to serve orders from any restaurant; in practice, some couriers may be willing to operate in broader areas, and so it might be possible to attain even better service quality times by exploiting that fact. Another extension of this framework relies in the incorporation of more complex behavioral aspects into the modeling of couriers, especially considering their level of autonomy; this may have important performance repercussions in the operation of meal delivery systems and to the best of our knowledge, it is a topic yet to be studied. Lastly, an alternative research direction involves

the use of learning-based methods to perform dynamic reshaping of regions. An interesting alternative to using a rolling horizon algorithm is to train a deep neural network to construct a region resizing policy via deep reinforcement learning techniques; then for a given state of the system, the resulting policy would dictate the modifications to be performed to each of the courier regions. Such an approach might be able to better capture stochastic aspects of meal delivery operations in the decision making process, possibly achieving superior levels of courier and diner satisfactions.

CHAPTER 4

DYNAMIC COURIER CAPACITY ACQUISITION IN RAPID DELIVERY SYSTEMS: A DEEP Q-LEARNING APPROACH

4.1 Introduction

In this chapter, we focus on the problem of dynamically adjusting the delivery capacity of a meal delivery system based on demand signals. We view and model this problem as a sequence of *real-time* opportunities to decide whether or not to enlarge the pool of delivery resources, *i.e.*, exploiting the option to add couriers (for different lengths of time). Due to the inherent uncertainty of this setting even evaluating the impact of a decision is extremely difficult. In practice, delivery companies typically adjust their delivery capacity based on the *order-per-courier ratio* (OPC), a workload metric that divides the number of active orders by the number of on-duty couriers at a given time. Although simple to compute, making a capacity decision solely on the OPC leaves out aspects of the system operations that, if exploited, could lead to significantly better solutions. Therefore, we propose and explore the use of deep Q-learning, a reinforcement learning (RL) technique that approximates the value of state-action pairs using a neural network, in this context usually referred to as a deep Q-network (DQN). This methodology is well suited for the studied environment as it can be used to make real-time decisions (*i.e.*, the DQN can be trained offline and then queried in real time) and is able to factor in the future impact of current actions as well as system state features such as time of the day, order placement rate, number of available couriers, and order delivery promises. These characteristics make this type of methodology attractive for real-time decision-making in complex logistics systems.

4.1.1 Main contributions

The main contributions of this research can be summarized as follows:

- We present, to the best of our knowledge, the first study looking at dynamic fleet sizing in rapid delivery operations, and one of the first to use deep Q-learning for a dynamic transportation problem.
- We propose a deep RL framework to devise a policy to decide whether or not to add delivery capacity, and, if so, what type of delivery capacity, in an environment with uncertain, highly fluctuating demand considering order volume, order urgency, and active delivery capacity.
- We demonstrate the effectiveness of the devised policy by performing a series of experiments in which we compare the policy's performance against the performance of policies representing current practice.
- We analyze the sensitivity of the performance of learned policy to algorithmic configuration decisions and context-specific settings.

The remainder of the chapter is organized as follows. Section 4.2 provides a brief survey of the related literature. Section 4.3 defines and formulates the problem, and then introduces the RL-based solution approach. Section 4.4 provides empirical evidence of the effectiveness of the solution approach. Section 4.5 summarizes the work and suggests future research directions.

4.2 Relevant literature

Rapid delivery problems can be classified as dynamic vehicle routing problems as routing decisions are made as new orders are placed. The papers by [44] and [45] provide excellent reviews of this class of problems. More precisely, meal delivery operations fall under the

scope of dynamic pickup and delivery problems (dPDP), where couriers are dynamically assigned to retrieve orders from different vendors and drop them off at customer locations. The related literature in this area is vast, and a thorough survey on dPDP is provided by [10].

The existing work on dPDP covers a wide range of applications that include general same-day delivery [6, 23, 34, 62, 64, 68], ride-sharing [3, 20, 70], and meal delivery [47, 66, 73]. However, most of that research focuses on routing decisions assuming a given set of courier blocks (*i.e.*, known delivery capacity) for the entire operating period. [8] and [9] study versions of the fleet-sizing problem in the contexts of meal delivery and ride-sharing, respectively, although in settings where information about order arrivals is known in advance and these papers do not consider dynamically expanding delivery capacity. In contrast, here we specifically focus on dynamically expanding delivery capacity based on observed order arrivals, *i.e.*, augmenting the base courier fleet when demand is higher than expected, a problem that to date has not been explored in dynamic delivery settings.

A related line of research can be found in the vehicle scheduling literature, which studies problems in which one seeks an optimal number of vehicles to serve a set of requests. A comprehensive survey on this line of research can be found in [15]. The most basic setting of the vehicle scheduling problem, the deterministic single-depot variant, was first solved by [49]. This single-depot variant can by solved in polynomial time, whereas the multi-depot extension is NP-hard [11]. [30] considers a setting where travel times are not deterministic, potentially causing delays in the start time of serving requests; the authors propose to solve a sequence of optimization problems, each considering multiple scenarios for future travel times. Other authors further extend this setting by considering other potential disruptions (*e.g.*, vehicle breakdowns and traffic congestion) and evaluate the benefit of corrective actions that modify the schedules of the vehicles [26, 59]. However, none of this research considers an environment in which requests arrive in real-time and information

about a request is revealed only when it is placed, as is the case in the environment studied in this chapter.

Our methodological approach is based on deep Q-learning, a reinforcement learning algorithm first proposed by [40]. Adoption of deep learning and Q-learning techniques in the field of transportation has recently been on the rise, but there are significant opportunities for further research in the use of these methods. [46] address the problem of trip-driver assignments in ride-hailing contexts. They improve the traditional linear assignment model by refining the matching weights using CVNet, a driver-decentralized deep Q-network (DQN) that estimates the long-term value of each feasible assignment. The authors also describe the deployment process of the algorithm for the ride-hailing company DiDi Chuxing and report significant benefits in multiple business metrics when compared to the traditional linear optimization model. [19] use deep-Q learning to solve a single depot dynamic delivery routing problem using a fleet comprising vehicles and drones. A DQN is employed to determine whether or not an order is served and if so, whether it is served by a vehicle or a drone. The authors make routing decisions using heuristic methods. [36] explore the effectiveness of DQN and actor-critic algorithms in learning policies for fleet management, via courier repositioning in online ride-hailing operations. The authors make use of contextual information to explicitly incorporate coordination between numerous drivers in the learning process and explore the effect of multiple modeling and algorithmic choices.

The use of machine learning methodologies in logistics is still in an early stage of development, but may offer great potential for improving scalability and for solving more complex problems [17, 71], and the research results we present in this chapter offer another example of this potential.

4.3 Methodology

4.3.1 Problem description

Let $\mathcal{H} = [0, H]$ be an operating period for a rapid delivery system (and $H > 0$ the horizon), typically representing a day of operations. At different times during the operating period, delivery orders are placed at one of N depots. For each order o, let t_o be the time when it is placed. Information about an order o becomes known only at t_o, *i.e.*, its pickup and dropoff locations p_o and d_o, respectively, and its ready time e_o at the pickup location. When an order is placed, the customer placing the order receives a promised delivery time m_o; if the order cannot be delivered by this promised time, the order is assumed lost. No orders are placed after time $H_0 < H$.

Orders are continuously assigned to on-duty couriers to be delivered at their dropoff location. Each courier has a start time $t_{start} \in \mathcal{H}$ and a number of working hours $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of working period lengths for couriers; namely each courier may be assigned to deliver new orders only after t_{start}, and the courier is no longer able to accept new orders by $\min\{H, t_{start} + 60c\}$ (in the sequel, we refer to couriers that work for $c \in \mathcal{C}$ hours as *c-hour couriers*). At any time $t \in \boldsymbol{T}$, unassigned orders are sequentially assigned to available couriers in a greedy manner based on the remaining time until the order becomes lost, giving first priority to orders closer to their due time. A courier can only be assigned to deliver a given order o if (i) the resulting order *pickup* time is no later than the end of the courier working period (a courier may still complete its last delivery later than the end of its working period); and (ii) the resulting order *delivery* time is no later than the delivery time promise m_o. Consequently, an order o becomes lost if it is still unassigned at its latest pickup time, *i.e.*, the pickup time that results in the order being delivered exactly at its due time, m_o.

If a courier q arrives at p_o at time t to pick up order o, we assume q picks it up at a time

$t_{pickup} = \max\{r_o, t + s_p\}$, where s_p is the time it takes q to walk from its vehicle to the depot; and we further assume that q starts driving towards the dropoff location d_o at time $t_{pickup} + s_p$. Likewise, when q arrives at dropoff location d_o at time t', we assume that o is effectively delivered at time $t_{dropoff} = t' + s_d$, where s_d is the time it takes q to reach the customer and handover the order; and that q resumes its duties at time $t_{dropoff} + s_d$. At time $t_{dropoff} + s_d$, if q has already been assigned another order, the courier immediately starts heading towards that order's pickup location. Otherwise, we assume that q starts repositioning towards the closest depot from its current location. However, if at any point during the repositioning q is assigned to a new order, the courier immediately starts moving towards the order's (possible different) pickup location.

Prior to the start of the operating period, a set of base couriers is scheduled for the day based on expected demand information (expected order volume and placement times). For simplicity, the start location of each base courier is assumed to be the centroid of the depot locations. However, the inherent uncertainty of order placements might result in the system's delivery capacity being surpassed by the order volume during the day of operation, at which point the decision maker may request and receive additional on-demand couriers to ease the workload. In the remainder, we will refer to the decision maker as the *agent*. The start location of on-demand couriers added at time t is assumed to be the depot with largest number of active unassigned orders at time t. Moreover, at time t, if a decision is made to add on-demand couriers, then the effective start time of these couriers is $t + \delta$, where δ is a fixed show-up delay. On-demand couriers will also have a planned working duration and specific cost required to work that duration. The type c of an on-demand courier specifies these parameters, and we assume that one or more types of on-demand couriers are available to be added.

The agent's decision problem is then to decide when and how many on-demand couriers to add with the objective of minimizing the expected overall system cost, which captures the

cost of missing orders' delivery time promises, and the cost of dynamically enlarging the system's delivery capacity via on-demand couriers.

4.3.2 Markov decision process formulation

Next, we formulate the problem as a Markov decision process. This formulation is at the core of our deep Q-learning framework.

Decision epoch. A time at which the agent decides whether or not to expand delivery capacity by adding one or more on-demand couriers. We assume the agent makes a decision every Δ time units and that the last epoch occurs at H_0. Hence, the set of decision epochs is $T_{action}(\Delta) \doteq \{i\Delta : i \in \{0, 1, \ldots, \lceil \frac{H_o}{\Delta} \rceil\} \subseteq \boldsymbol{T}$.

State. The state at a epoch $t \in T_{action}(\Delta)$, denoted by s_t, encodes the features of the system that we believe are relevant to make a decision. The features include:

- the time remaining until the end of the operating period, *i.e.*, $H - t$, to capture anticipated temporal demand patterns during the operating period.
- the number of active couriers, $q_t^{courier}$, *i.e.*, the number of couriers currently on-duty plus the number of couriers scheduled to start at time t.
- the number of active orders in the system, q_t^{orders}, *i.e.*, the number of placed orders that have not been delivered.
- a j_1-dimensional vector Θ_t^1 encoding the scheduled changes in the number of on-duty couriers during the time windows $\{(t, t+k], (t+k, t+2k], \ldots, (t+(j_1-1)k, t+j_1k]\}$, which captures (allows computing) the available delivery capacity in the near future.
- a j_2-dimensional vector Θ_t^2 encoding the number of orders placed during the time windows $\{(t - j_2k', t - (j_2 - 1)k'], \ldots, (t - 2k', t - k'], (t - k', t]\}$, which captures (allows predicting) the workload in the near future (which, in turn, allows predicting

whether additional delivery capacity will be needed). If for some $j \leq j_3$ we have $t-(j-1)k'' < 0$, then the count corresponding to $(t-jk'', t-(j-1)k'']$ is set to 0.

- a j_3-dimensional vector Θ_t^3 encoding the number of orders that will become late during time windows $\{(t, t+k''], (t+k'', t+2k''], \ldots (t+(j_3-1)k'', t+j_3k'']\}$ if no additional delivery capacity is added, which captures (allows determining) the near-term delivery capacity needs.

Thus, at time $t \in \boldsymbol{T}$, we represent the state as $s_t = (H-t, q_t^{couriers}, q_t^{orders}, \Theta_t^1, \Theta_t^2, \Theta_t^3)$.

Action. At decision epoch t, the agent chooses an action a_t which specifies for each working period length $c \in \mathcal{C}$ how many couriers with that working period length to add, with the restriction that at most $\overline{m}_c$ couriers of type c can be added (not adding any courier is also a possible action). Any on-demand couriers added at t enter the system at time $t+\delta$, where $\delta > 0$ is a delay (in minutes).

Reward. Adding an on-demand c-type courier results in a negative reward $K_c < 0$, and a lost order results in a (negative) reward $K_{lost} < 0$. An order is lost as soon as it can be determined that it cannot be delivered on time, and we ignore it from that point on. (At time t, determining whether an order is lost takes into account any couriers added at time t.) Let n_{t_1,t_2} be the number of orders lost during $[t_1, t_2)$, and let t^+ denote the time of the decision epoch following the decision epoch at time t. Then an action a_t at time t that adds k_t^c on-demand couriers of type c for $c \in \mathcal{C}$, results in an *immediate* reward at time t of $r_t(s_t, a_t) = K_{lost} \cdot n_{t+\delta,t^+ +\delta} + \sum_{c\in\mathcal{C}} K_c \cdot k_t^c$. Observe that the lost orders associated with action a_t do not include orders that are lost prior $t+\delta$, the start time of any on-demand couriers added at time t, as these on-demand couriers cannot prevent orders being lost before $t+\delta$.

Transition. The state is periodically updated to reflect decisions at epoch $t \in T_{action}(\Delta)$, any order placements since the last update, and any order – courier assignments since the

last update.

- At time t, $q_t^{couriers}$ is updated to reflect any couriers ending their duty and any couriers starting their duty at t, and q_t^{orders} to reflect any orders placed and any orders delivered at t. Likewise, the corresponding counts in Θ_t^1, Θ_t^2 and Θ_t^3 are updated accordingly.

- If action a_t implies adding on-demand couriers, then Θ_t^1 is updated accordingly, by modifying the change in the number couriers to occur for the time window containing $t + \delta$.

- If an order is placed and it is not immediately assigned to a courier, the order increases the count of one entry of Θ_t^3, since the order may become late in the future.

- Given the assignment logic, once an order is assigned to a courier, it is no longer at risk of being lost and therefore the late order count in Θ_t^3 is updated correspondingly.

Objective. Let Π be the space of policies π that map each possible state to a feasible action. The optimal policy is then specified by

$$\pi^* = \underset{\pi \in \Pi}{\arg\max}\ \mathbb{E}\left[\sum_{t \in T_{action}(\Delta)} r_t(s_t, a_t) \middle| s_0\right]$$

namely, a policy that maximizes the total expected reward over the decision horizon.

4.3.3 Deep Q-learning

Formulating the problem as a Markov decision process allows it, in theory, to be solved using the Bellman equation (4.1) and backward recursion, where $V(s_t)$ is the value function of state s_t, and A_t is the action space at time $t \in T_{action}(\Delta)$.

$$V(s_t) = \max_{a \in A_t} \left\{ r(s_t, a) + \mathbb{E}[V(s_{t+\Delta}) | s_t] \right\}, \qquad \forall t \in T_{action}(\Delta) \tag{4.1}$$

However, the size of the state space in this problem is too large to enumerate. Furthermore, even trying to approximate the value function using tabular RL methods (*e.g.*, tabular Q-learning) is not practically feasible, as storing the value of every possible state-action pair incurs prohibitive memory consumption and accurately learning the value of every pair would require a massive exploration in order to observe all possible state-action combinations. By contrast, deep Q-learning allows to mitigate these practical issues since neural networks can learn the relationship between different state and actions, and use this information to extrapolate the value from explored state-action pairs to the value of unexplored pairs.

The Q-learning algorithm learns the value of taking an action at each given state. The so-called *Q-value* of a state-action pair is learned by a neural network, the DQN. Let θ be the current weights characterizing the DQN that models the agent, and for a state-action pair (s_t, a_t), let $Q_\theta(s_t, a_t)$ be the Q-value that the DQN of weights θ associates with that state-action pair. The Q-values approximate Equation (4.1) as follows

$$V(s_t) \approx \max_{a \in A_t} Q_\theta(s_t, a). \tag{4.2}$$

Moreover, by performing the learning phase of the DQN offline and storing only its weights θ, the agent is later able to quickly compute, for any state s_t, the Q-value $Q_\theta(s_t, a_t)$ for every possible action a_t, thus easily identifying the action with the largest expected reward. At decision epoch $t \in T_{action}(\Delta)$, the action a_t selected by θ to be executed corresponds to the one that maximizes the future expected rewards, *i.e.*,

$$a_t = \arg\max_{a \in A_t} Q_\theta(s_t, a), \quad \forall t \in T_{action}(\Delta) \tag{4.3}$$

DQN architecture. We model the agent as a multi-layer perceptron, a fully connected neural network characterized by an input layer; u_{layers} hidden layers, each with u_{nodes}

nodes, and an output layer. Given an observed state s_t, the input layer is fed with the encoded information and is passed to the nodes in the first hidden layer. Each hidden layer receives the output from the previous layer, where each of the u_{nodes} nodes in the layer first computes the dot product between the weights of the inbound connections and the values passed through each connection from the previous layer, and then an activation function is applied on the resulting value before it is passed to each node of the next layer. Our model uses the rectified linear unit function (ReLU) as the activation function for each hidden layer. The last hidden layer passes its output to the output layer, which estimates, for each action $a \in A_t$, the corresponding Q-value $Q_\theta(s_t, a)$.

Training settings. The weights θ of the DQN are learned in a training phase. This phase consists of simulating a total of $N_{episodes}$ instances (simulating an instance is referred to as an episode), each representing a day of operations, *i.e.*, a realization of order placements and a schedule of base couriers.

To make the agent more robust and able to handle multiple scenarios, at the beginning of an episode a daily order placement profile p describing the order placement rate throughout the day is randomly sampled from a set of possible daily order profiles P; then, the number of orders placed during the day and their placement times are sampled from the sampled pattern p. During an episode, the agent evaluates the system state s_t at every decision epoch $t \in T_{action}(\Delta)$ using the current weights θ, and chooses an action a_t, *i.e.*, whether or not to increase the delivery capacity and if yes, how to do so. Given the action, it then collects the associated reward $r_t(s_t, a_t)$ and reaches the next decision epoch at a post-decision state s'_t (see Figure 4.1 for an illustration of this process). Once the post-decision state is observed, an *experience tuple* $(s_t, a_t, r_t(s_t, a_t), s'_t)$ is stored to later be used in the update of the DQN parameters.

In the training phase, at each decision epoch t, the agent typically selects the action a_t that maximizes the total expected reward associated with the current state s_t (see Equation

(4.3)). This greedy approach is known as *exploitation*. However, training solely using exploitation may prevent the agent from observing most of the state-action space, thus not being able to find potentially better actions for given states and ultimately getting stuck at a local optima. To mitigate this effect, the agent is occasionally forced to take a random action; this is known as *exploration*. In particular, we adopt an ε-greedy exploration policy during the training phase: at every decision epoch t the agent selects an action uniform randomly from the action space A_t with probability ε, and selects an action based on Equation (4.3) with probability $1 - \varepsilon$. After completing an episode, the ε parameter is slightly decreased to gradually increase the likelihood of making decisions based on the Q-values estimated by the agent toward the end of the training phase.

At the end of an episode, instead of immediately training the DQN with the experience tuples just collected, these are first placed in a memory storage, and then the DQN weights θ are updated using a batch of b tuples randomly sampled from that storage, a process known as experience replay [37]. Experience replay makes use of a memory with finite storage capacity $M > b$; once the memory becomes full, newest stored experience tuples overwrite the oldest stored tuples. By using experience tuples from both present and past episodes, this practice seeks to train the DQN using less correlated observations, ultimately stabilizing the parameter updates and improving the training speed of the DQN.

Algorithm 3 illustrates the DQN training process. Line 1 initializes the DQN parameters θ and the exploration parameter ε. A set of preliminary episodes is run in order to completely initialize the memory with M experience tuples (lines 2 - 5). Then for each of the $N_{episode}$ episodes, the algorithm simulates a complete day of operations gathers $|T_{action}|$ experience tuples and stores them in the experience replay memory, overwriting the oldest stored tuples (lines 7 and 9). At the end of the episode, line 10 samples a batch of size b uniformly at random from the memory. For each sampled tuple, a target Q-value $Q_{\theta,i}^{target}$ is computed, assuming for non-terminal post-decision states s_i' that the best action is the one that maxi-

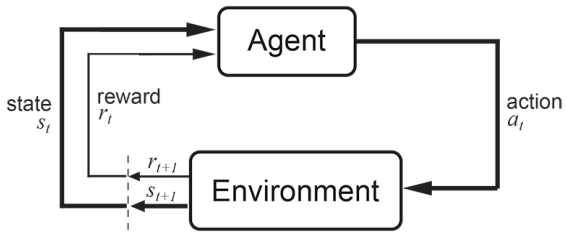


Figure 4.1: Standard training process in a RL setting [58]

mizes the expected total reward given that the system is in state s_i', and for terminal states s_i' that the reward is simply r_i (lines 11 - 15). Consequently, the algorithm computes the total loss $L(\theta)$ comparing the target values $Q_{\theta,i}^{target}$ with the observed Q-values $Q_\theta(s_t, a_t)$ in line 16, and the loss is then used by the optimizer method O to update the DQN weights θ in line 17. After updating θ, the algorithm updates the exploration parameter ε in line 18. Once all the episodes have been simulated, the algorithm returns the trained DQN weights.

Let θ^* be the DQN weights trained by Algorithm 3. Then for a given state s_t, our deep Q-learning policy selects the action a that maximizes the Q-value $Q_{\theta^*}(s_t, a)$, as in Equation (4.3). We denote this policy as π^{DQN}.

4.3.4 Baseline policies

To assess the performance of the learned policy π^{DQN}, we propose a set of baseline policies for comparison. These policies react only to the observed OPC, with the goal of creating a policy that mimics how on-demand courier addition decisions are made in practice. We then use these baseline policies as strawmen to benchmark against π^{DQN}.

The simple baseline policies compare an observed value of OPC to a threshold value, and if the observed value exceeds the threshold then on-demand couriers are added until a target OPC is achieved. Let $opc(t)$ be the observed OPC level at time $t \in \boldsymbol{T}$, opc_{max} be the maximum allowed OPC level, and opc_{target} a target OPC level ($opc_{target} \leq opc_{max}$). Then for each type of courier $c \in \mathcal{C}$, we define the base policy $\pi^c(opc_{max}, opc_{target})$ as follows:

Algorithm 3 DQN Training Phase

Input: Loss function L, discount factor γ, initial exploration rate ε_0, minimum exploration rate ε_{min}, epsilon decay ε_{decay}, number of episodes $N_{episodes}$, set of decision epochs T_{action}, action space A_t, batch size b, initial DQN weights θ_0, memory size M, optimizer method O.

Output: Trained DQN weights θ.

$\theta \leftarrow \theta_0, \varepsilon \leftarrow \varepsilon_0$
for $e \in \left\{1, \ldots, \left\lceil \frac{M}{|T_{action}|} \right\rceil \right\}$ **do**
    for $t \in T_{action}$ **do**
        Select an action $a_t \in A_t$ using the current ε-greedy policy.
        Observe and store the experience tuple (s_t, a_t, r_t, s'_t) into the experience replay memory.
for $e \in \{1, \ldots, N_{episodes}\}$ **do**
    for $t \in T_{action}$ **do**
        Select an action $a_t \in A_t$ using the current ε-greedy policy.
        Observe and store the experience tuple (s_t, a_t, r_t, s'_t) into the experience replay memory.
    Collect a uniformly-random sample a batch of b experience tuples (s, a, r, s') from the memory.
    for $i \in \{1, \ldots, b\}$ **do**
        if s'_i is terminal **then**
            $Q_{\theta,i}^{target} = r_i$
        else
            $Q_{\theta,i}^{target} = r_i + \gamma \max_a Q_\theta(s'_i, a)$
    Compute loss $L(\theta)$
    Update θ using optimizer O and total loss $L(\theta)$.
    $\varepsilon \leftarrow \max\{\varepsilon - \varepsilon_{decay}, \varepsilon_{min}\}$
return θ

at every decision epoch $t \in T_{action}(\Delta)$ (and *right after* the couriers scheduled to start and end their block at t do so) we observe $s_t = opc(t)$ and check if it exceeds opc_{max}. If that is the case, then $\pi^c(opc_{max}, opc_{target})$ performs the action a_t that adds an extra number m_t^c of couriers of type c to the system until $opc(t)$ falls below opc_{target}. More precisely, if $o(t)$ and $z(t)$ are the number of active orders and couriers in the system at time t, respectively, then the number of couriers added by $\pi^c(opc_{max}, opc_{target})$, is

$$m_t^c = \begin{cases} 0 & \text{if } opc(t) \leq opc_{max} \\ \lceil \frac{o(t)}{opc_{target}} \rceil - z(t) & \text{otherwise} \end{cases}$$

and $opc(t)$ is updated as

$$opc(t) = \frac{o(t)}{z(t)+m_t^c}.$$

4.3.5 Performance metrics

We measure the performance of each considered policy using three key metrics, namely (i) the fraction of orders delivered (on-time) on a day, which represents the service level attained by the policy; (ii) the total number of courier-hours added during a day, which measures the amount of additional resources required by the policy to achieve its service level; and (iii) the total reward (without discount) over a day, which captures the trade-off between customer service and the cost of enlarging the capacity. More precisely, let $(\bar{s}_t, \bar{a}_t)$ be the state-action tuples observed by the agent at each decision epoch $t \in T_{action}(\Delta)$ of a given run, with $\bar{s}_t$ being the observed state at t (for π^{DQN}, the state s_t is defined as in Section 4.3.2; for π^c it is simply $opc(t)$; and for π^{hybrid} it is t and $opc(t)$), let $\bar{a}_t$ the corresponding action taken; and let n_0^H be the number of orders lost during $\mathcal{H}$ of a total of n placed orders. Then the total reward is computed as $\sum_{t \in T_{action}(\Delta)} r_t(\bar{s}_t, \bar{a}_t)$, and the

service level corresponds to $\frac{n-n_0^H}{n}$.

4.4 Experimental results

This section shows the evolution of the learning that occurs during the training of π^{DQN}, and presents the results of a comparison between the performance of the learned policy π^{DQN} and two baseline policies in terms of total reward and service quality.

4.4.1 Experimental settings

Each episode covers an operating period of $H = 540$ minutes during which orders are dynamically placed at a set of $N = 16$ depots. Orders are placed during the first $H_0 = 450$ minutes. Service times are set to $s_p = s_d = 4$ minutes. For an order o, the ready time is set to $e_o = t_o + 10$ and the promised delivery time is set to $m_o = t_o + 40$.

At the beginning of each episode, the number of orders placed at each depot d, n_d, is drawn from a uniform distribution, *i.e.*, $n_d \sim U[10, 20]$ for $d \in \{1, \ldots, N\}$; the total number of orders, n, therefore, is $n = \sum_{d=1}^{N} n_d$. An order placement time pattern p is then uniformly sampled from a set of patterns P comprising 4 different scenarios; a pattern p corresponds to a probability mass function as depicted in Figure 4.2a. Given a selected pattern, each order's placement time is sampled using the mass function for pattern p. Each of the scenarios in P has a moderate peak in order placements around lunch time, a large peak in order placements around dinner time, and an off-peak period with relatively few order placements between the two peaks, reflecting what is observed in meal delivery operations in practice. However, there are differences between the four scenarios that reflect external conditions that impact demand, such as special events or changes in weather.

Both base and on-demand couriers work for shifts of either 1 or 2 hours in duration (*i.e.*, $\mathcal{C} = \{1, 2\}$), so we model each action as a two-dimensional vector (a_t^1, a_t^2) where a_t^c is the number of c-hour couriers added at time t. Moreover, there is a decision epoch, a time at

which the agent takes an action, every $\Delta = 5$ minutes. At each decision epoch, we also limit the agent to add up to 2 on-demand 1-hour couriers and up to 1 on-demand 2-hour courier, *i.e.*, $A_t = \{0, 1, 2\} \times \{0, 1\}$.

Base capacity schedule. The number of base c-hour couriers, D_c for $c \in \mathcal{C}$, to be scheduled at the beginning of an episode is randomly sampled from a discrete uniform distribution, with $D_1 \sim \mathcal{U}[30, 40]$ and $D_2 \sim \mathcal{U}[20, 30]$. The scheduling is based on the average order placement profile $\overline{p}$ shown in Figure 4.2b (obtained by averaging the placement patterns in P). The start times of the couriers are set as follows:

- For the 1-hour couriers, $\lceil \frac{D_1}{6} \rceil$ of them are set to start at time 60, and another $\lceil \frac{D_1}{6} \rceil$ couriers to start at time 120. Furthermore, another $\lceil \frac{D_1}{3} \rceil$ couriers are set to start at time 270, and the remaining couriers are set to start at time 330.

- For the 2-hour couriers, $\lceil \frac{D_2}{6} \rceil$ of them are set to start at time 0, and another $\lceil \frac{D_2}{6} \rceil$ couriers to start at 120. Furthermore, another $\lceil \frac{D_2}{3} \rceil$ couriers are set to start at time 240, and the remaining couriers are set to start their working period at 360.

- If it is a training episode, then each courier's start time is perturbed by adding a realization of the discrete uniform random variable $u \sim \mathcal{U}[-20, 20]$ (a perturbed start time that occurs before the beginning of the horizon, namely a negative value, is replaced by 0).

The reason for having a random number of base couriers and perturbed start times of the base couriers is to facilitate exploration of more state-action pairs during training (by having scenarios in which the system is under-staffed and over-staffed) so that the policy learned by the agent is able to take good decisions in variety of settings.

Training settings. We model the DQN as a multilayer perceptron (*i.e.*, a fully-connected neural network) with n_{hidden} hidden layers, each with n_{nodes} nodes and ReLU activation

Table 4.1: Tested and selected values for training parameters. Final values are selected based on best average reward obtained in the testing phase.

Parameter	Tested values	Selected value
$N_{episodes}$	$\{1, 2, 3, 5, 10\} \cdot 10^4$	$3 \cdot 10^4$
n_{layers}	$\{1, 2, 3\}$	3
n_{nodes}	$\{3, 4, 5, 6\} \cdot 10$	60

function, and an output layer with $|A_t|$ nodes. The results we present in this section correspond to a policy trained with the selected values specified in Table 4.1, which were selected based on the average reward obtained by the corresponding trained policy over the test instances. Regarding the number of episodes used for training, $N_{episodes}$, we do not observe any significant improvement when training the DQN for more than 30,000 episodes for the tested values of n_{layers} and n_{nodes}, hence we set $N_{episodes}$ to that value to keep the duration of the training phase to a minimum.

At a given decision epoch t, the components $\Theta_t^1, \Theta_t^2, \Theta_3^t$ of the state vector s_t are defined such that $(j_1, k) = (4, 30), (j_2, k') = (6, 5)$, and $(j_3, k'') = (2, 20)$. That is, the scheduled changes to the pool of on-duty couriers after time t, Θ_t^1, are captured using time windows $\{(t, t + 30], (t + 30, t + 60], (t + 60, t + 90], (t + 90, t + 120]\}$, which are the most relevant windows for an on-demand courier scheduled at t since a courier works for up to 2 hours in these experiments; the number of placed orders before time t, Θ_t^2, are captured using time windows $\{(t - 30, t - 25], (t - 25, t - 20], \ldots, (t - 10, t - 5], (t - 5, t]\}$, which are the most relevant given that the delivery time promise is 40 minutes; and the number of unassigned orders at time t, Θ_t^3, whose latest pickup time falls in the time windows $\{(t, t + 20], (t + 20, t + 40]\}$, which covers all active orders since none of them may spend more than 40 minutes unassigned before being lost. Prior to feeding s_t as input to the DQN, the encoded information is normalized using a min-max scaler to give more accurate relative importance between the considered features [12].

Given that the agent takes action a_t, on-demand couriers associated with a_t start their work-

ing period after a delay of $\delta = 5$ minutes (*i.e.*, they start at $t+5$). The unit reward of adding a courier of each type are $(K_1, K_2) = (-0.5, -0.8)$. Note that the per-hour cost of a 2-hour courier is less than the per-hour cost of a 1-hour courier; this is justified by the fact that 2-hour couriers have more flexibility to receive orders due to the assignment feasibility conditions, thus having the potential of serving more orders. The unit reward for lost order is $K_{lost} = -2$ so service quality degradation receives a heavier penalization than adding an extra unit of courier capacity. We set discount factor employed in our experiments to $\gamma = 0.99$.

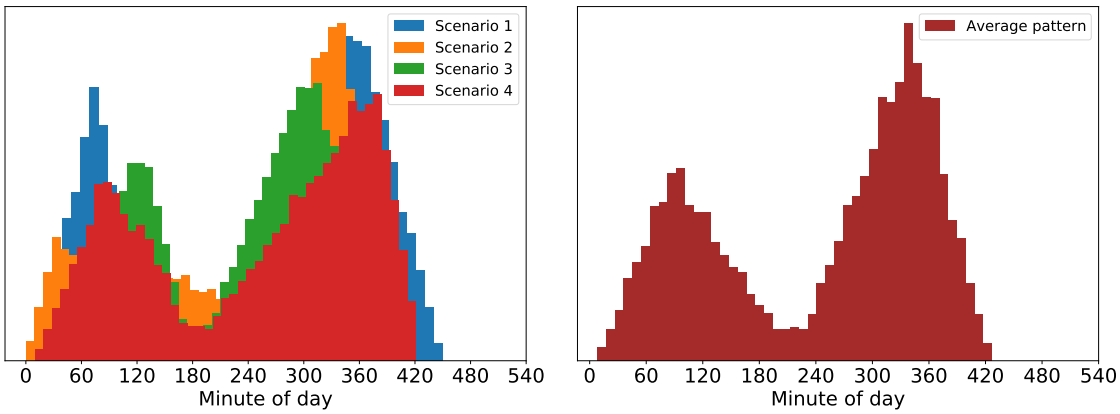
We train the agent using an ϵ-greedy approach to effectively explore the state space during training, starting with a value of $\varepsilon_0 = 1$ and linearly decreasing it to $\varepsilon_{min} = 0.01$ over the considered $N_{episodes}$ episodes, thus $\varepsilon_{decay} = \frac{1-0.01}{30000}$. The loss function we consider to train the DQN is the *mean-square error*, and the optimization of the DQN weights θ is performed using the Adam optimizer [32], with a learning rate of 10^{-3}, and parameter values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\varepsilon = 10^{-3}$. Our implementation of experience replay considers a memory size of $M = 10,000$ tuples, and after each episode the network weights are trained using a random minibatch of size $b = 1,000$.

4.4.2 Learning π^{DQN}

Next, we investigate how the performance achieved by the policy during training improves over time, and the corresponding changes in the number of added on-demand couriers of each type.

Figure 4.3 illustrates the evolution of the average total rewards, the average service level, and the average number of added on-demand 1-hour and 2-hour couriers, as the agent learns π^{DQN}. In particular, each data point is the average of 100 adjacent episodes.

At the beginning of the training phase, the agent follows a policy heavily focused on exploration (*i.e.*, high frequency of random actions in the $\varepsilon-$greedy exploration) that is able to



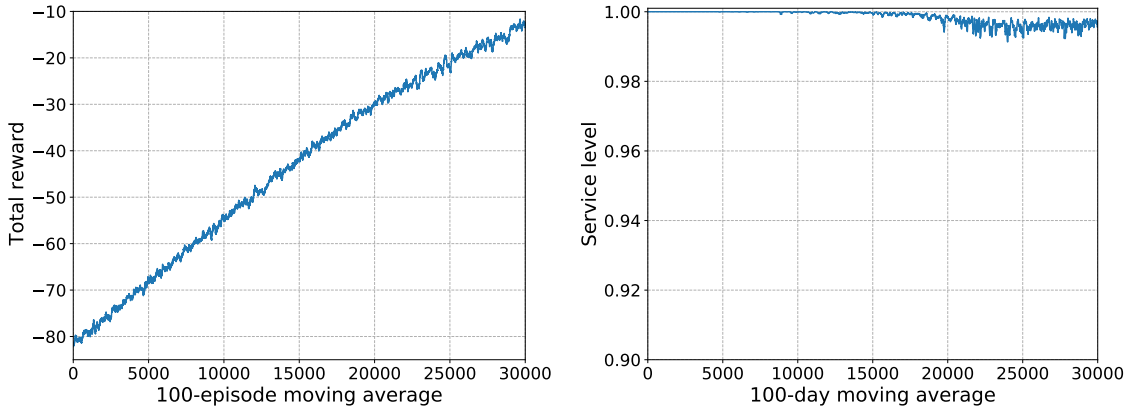
(a) Graphical representation of the set of order placement patterns P. Each episode samples its demand realization from one of these patterns

(b) Average order placement pattern obtained from the patterns in P, used to schedule base courier capacity

Figure 4.2: Evolution of performance metrics during training episodes; 500-episode moving average

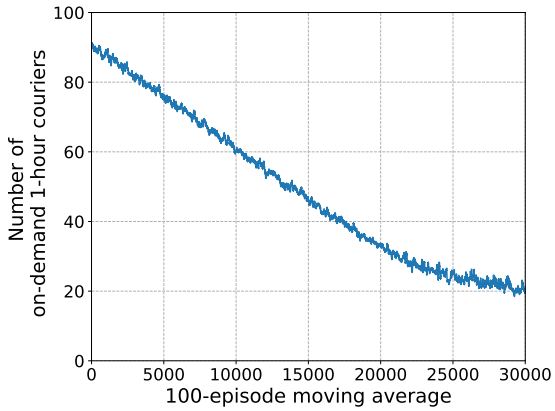
serve 100% of the orders, but at the expense of incorporating a large number of on-demand couriers as five out of the six possible actions result in adding extra delivery capacity to the system, thereby greatly increasing the total costs. However, we observe that the average reward attained by the agent steadily increases during the training phase, as illustrated in Figure 4.3a. We can also see the agent's transition from exploration to exploitation; after episode 20,000 the average reward improvement slows down due to a lower exploration rate ε.

This is also reflected in the evolution of the average service level and the average number of on-demand 1-hour and 2-hour couriers added, shown, respectively, in Figures 4.3b, 4.3c, and 4.3d. The number of 1-hour and 2-hour on-demand couriers added shows a steady decrease over the first 20,000 episodes. After this point, the number of on-demand 1-hour couriers added by the agent starts to converge, finally reaching an average of 20 1-hour couriers added per episode. Interestingly, the agent does not "appreciate" the lower per-hour cost of adding on-demand 2-hour couriers, adding at most one 2-hour courier per episode by the end of the training phase. This is likely due to the base fleet scheduling and the order placement patterns: in most situations, the periods during which extra couriers

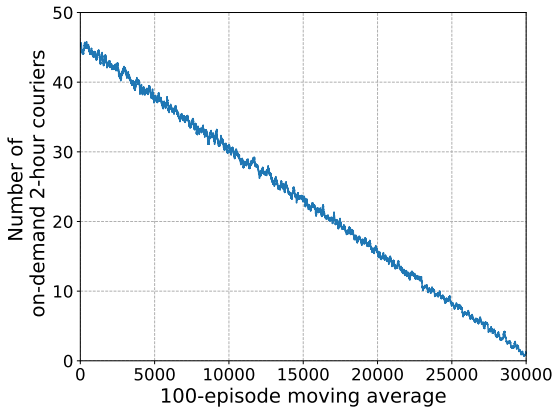


(a) Total rewards

(b) Service level



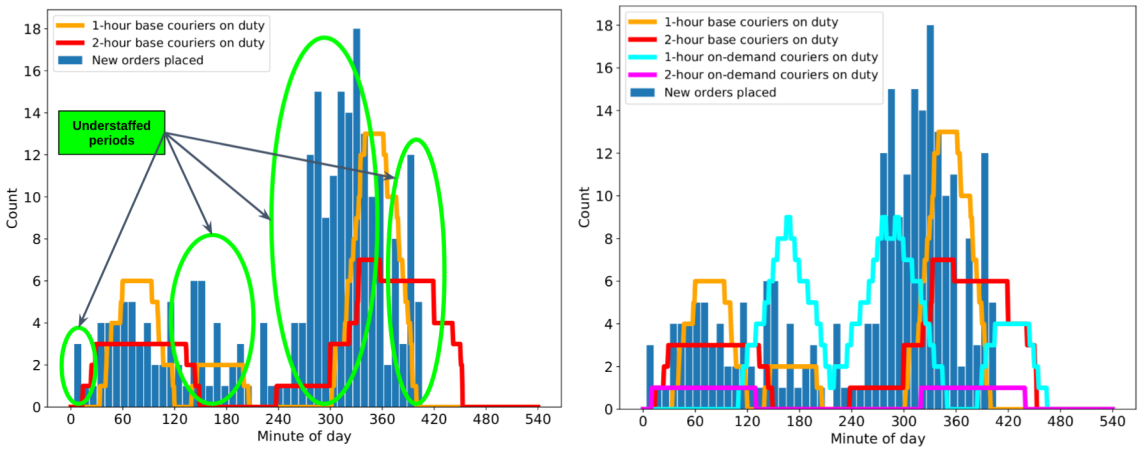
(c) 1-hour on-demand couriers



(d) 2-hour on-demand couriers

Figure 4.3: Evolution of performance metrics during training episodes; 100-episode moving average

are needed are relatively short, hence using the more expensive 2-hour couriers is not beneficial. We see that near episode 20,000 the service level slightly decreases as the number of on-demand couriers added reaches a level at which serving all orders is no longer trivial. Interestingly, the agent is in general able to identify market signals that indicate when additional courier capacity is required, thereby preventing a capacity shortage when demand deviates from the average order placement pattern, as illustrated in Figure 4.4. Ultimately, this allows an average reduction of 78% in the number of on-demand couriers added per episode by the end of the training phase (relative to the first set of training episodes) while successfully serving an average of over 99.5% of the orders.



(a) Order coverage *without* on-demand couriers (b) Order coverage *with* on-demand couriers

Figure 4.4: Order coverage in scenarios with and without the capability of adding on-demand courier capacity. Adding couriers in an on-demand basis makes possible to prevent capacity shortage when realized order placement deviates from the average pattern

Table 4.2: Tuned values of opc_{max} and opc_{target} for baseline policies

Policy	opc_{max}	opc_{target}
π^1	2.0	1.5
π^2	2.0	1.5

4.4.3 Performance

In this section we report the results benchmarking the trained policy π^{DQN} against the baseline policies π^1 and π^2. For this purpose, we produce 500 independent instances using the order placement profiles in P and report the results of each of the policies on these instances. We consider five configurations of each of the baseline policies, by setting the hyperparameter opc_{max} to every multiple of 0.5 in the interval $[1, 3]$ and setting the target OPC to $opc_{target} = opc_{max} - 0.5$, and report the results for the configuration that yields the best average reward. Table 4.2 shows for each baseline policy, the parameter configuration that yields the highest average reward.

We evaluate the performance of the policies using three metrics: the average total reward, the average service level, and the average number of on-demand courier hours added (with the average taken over the 500 instances). The results are shown in Figure 4.5 where each

Table 4.3: Per-policy average and standard deviation of performance metrics over the 500 test instances

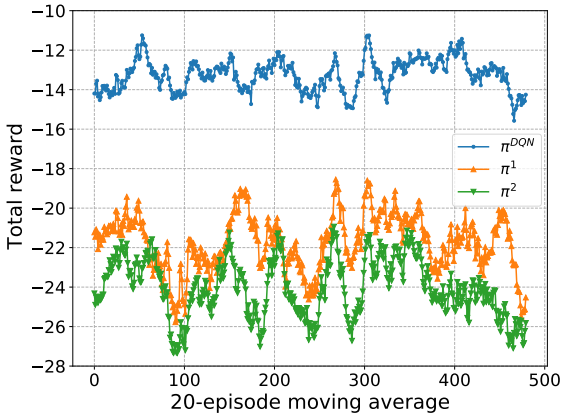
Policy	Service level		On-demand courier hours		Total reward	
	Average	Std. dev.	Average	Std. dev.	Average	Std. dev.
π^{DQN}	99.8%	0.34%	24.5	6.56	-13.2	4.12
π^1	98.1%	1.21%	25.0	8.04	-21.8	6.89
π^2	98.5%	1.01%	41.2	13.58	-24.1	6.96

curve depicts the evolution of the 20-instance moving average of the corresponding metric, *i.e.*, each data point represents the average of 20 instances (for a total of 480 data points). Furthermore, Table 4.3 reports the average and standard deviation of the values of the metrics for each policy over the 500 test instances.

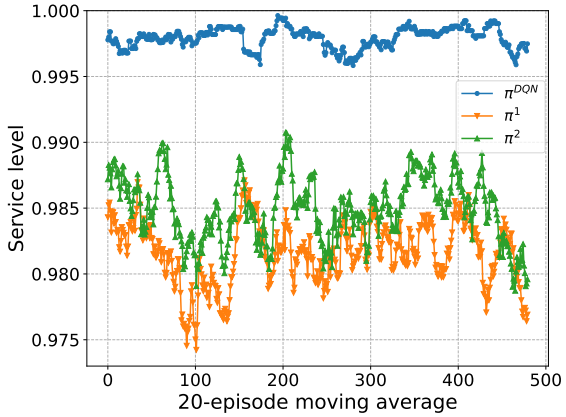
We observe that π^{DQN} greatly outperforms the policies π^1 and π^2 in terms of average reward: the learned policy not only improves the baselines rewards by about 40%, but is also more robust with a 40% lower standard deviation. The shows that π^{DQN} more accurately assesses when the system needs extra delivery capacity, which not only results in a better and more consistent service level, but does so with fewer on-demand couriers. This is especially clear when contrasting π^{DQN} with π^1: even though π^{DQN} almost exclusively adds 1-hour on-demand couriers (as does π^1), π^{DQN} significantly outperforms π^1 in terms of total reward adding about the same number of courier hours. The learned policy better exploits the information embedded in the state vector to properly time adding of on-demand capacity, thereby outperforming the purely OPC threshold-based policy.

4.5 Conclusion

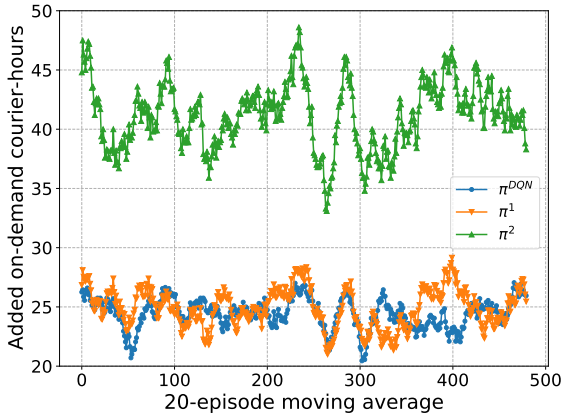
This chapter considers the application of deep RL techniques to study the problem of dynamically acquiring extra courier capacity in a meal delivery system. By definition, these systems are highly dynamic and present various sources of uncertainty, and hence solving any related problem results in a difficult task. Despite this, using DQN makes possible



(a) Total reward



(b) Service level



(c) On-demand courier hours

Figure 4.5: Policy benchmark over 500 test instances; 20-episode moving average

to incorporate many aspects of meal delivery in the capacity enlargement decision making, ultimately making possible to obtain a cost-efficient policy that outperforms standard practices in the industry.

The immediate next step is to develop a deeper understanding of the actions dictated by the obtained policy, specifically in terms of the courier type preference and the timing of the actions, and to consider improving the baseline policies to further justify the use of machine learning algorithms in solving this problem. Another key aspect to study is the robustness of the policy when applied to instances that significantly deviate from the ones used during training, with respect to both demand distribution and values of application-specific parameters.

In terms of the RL framework, it may be worth considering potential enhancements such as the use of target networks, double DQN and prioritized experience replay, which in various other settings have further improved the performance of the resulting policy. The positive results obtained in this chapter using deep Q-learning suggests that this methodology may be useful to study other aspects of meal delivery. In particular, a similar framework could be used to improve the results on dynamic zone management in Chapter 3, and to learn optimal courier-order assignment policies that optimize service quality, as in [47].

Appendices

APPENDIX A

ADDITIONAL MATERIAL FOR CHAPTER 2

A.1 Proofs

Proof of Proposition 1. If $|J| = 1$ the claim trivially follows and therefore we assume $|J| \geq 2$. Initially, let $J = J_1 \cup J_2$, $J_1, J_2 \neq \emptyset$ and $J_1 \cap J_2 = \emptyset$. Without loss of generality we can assume that $\tau_{J_1} \geq \tau_{J_2}$. Consider then a feasible schedule S_1 to Problem 1 that at time t dispatches two couriers c_1 and c_2 with order sets J_1 and J_2, respectively. Note that c_1 is unavailable for picking up other ready orders during time points $I_1 = \{t, t+1, \ldots, t+2\tau_{J_1}\}$, whereas c_2 will be unavailable to serve any new orders from t to $t + 2\tau_{J_2}$.

Alternatively, consider a schedule S_2 that bundles J into a single dispatch for courier c_1 at time t (which is possible since couriers do not have a fixed capacity). Since by definition $\tau_J = \tau_{J_1}$, dispatching c_1 also serves all $|J|$ orders during I_1 while courier c_2 remains at the depot beginning at time t, which is earlier than the return time $t + 2\tau_{J_2}$ in the above schedule. Thus, schedule S_2 dominates S_1. □

Proof of Lemma 2. Let $t' \in (t, r_{j+1})$, and consider schedule $S(t')$ that dispatches a courier at time t' with orders set $A(t')$. From $t \in [r_j, t')$ it follows that $A(t') \subseteq A(t)$: indeed, no new orders are placed in $(t, t']$ although some of the orders in $A(t)$ might not be active by t'. Hence, schedule $S(t')$ can always be improved by the one that moves the dispatch at t' to t, which is always possible by definition of t. □

Proof of Proposition 3. For $j \in N$, the possible times at which a courier might become available at the depot during $[r_j, r_{j+1})$ are r_j and any returning time $r_i + 2\sum_{k \in K} \tau_k \in$

(r_j, r_{j+1}) with $i < j$ and $K \subseteq N$ that results from a dispatch previous to r_j, thus Lemma 2 implies that an optimal schedule can be obtained by considering only such dispatch times.

Moreover, denoting the set of all dispatches of interest as $\mathcal{T}_0$, the only time points outside the depot to be considered are the potential delivery times $\{t + \tau_i : t \in \mathcal{T}_0, i \in A(t)\}$, at each of which a dispatched courier decides between traversing to a further delivery location or returning to the depot. □

Proof of Proposition 4. For a fixed binary vector $\overline{v}$ and $m \in \mathbb{Z}_+$, a feasible value of z must satisfy the constraints:

$$\overline{v}_{jp} \leq \sum_{q \in \delta_p^-} z_{qp}, \quad \forall j \in N, \quad \forall p \in \mathcal{V}_j \tag{A.1a}$$

$$\sum_{q \in \delta_\alpha^+} z_{\alpha q} = m \tag{A.1b}$$

$$\sum_{p \in \delta_\omega^-} z_{p\omega} = m \tag{A.1c}$$

$$\sum_{p \in \delta_q^-} z_{pq} = \sum_{r \in \delta_q^+} z_{qr}, \quad \forall q \in \mathcal{V} \setminus \{\alpha, \omega\} \tag{A.1d}$$

$$z_{pq} \in \begin{cases} \mathbb{R}_+ & \text{if } p, q \in \mathcal{V}_0, \\ \{0, 1\} & \text{otherwise} \end{cases} \quad \forall (p, q) \in \mathcal{A} \tag{A.1e}$$

By construction of the underlying time-expanded network, each non-depot node p has a unique arc $a_p \in \mathcal{A}$ inbound to p, thus the right hand side of (A.1a) can be written as $\sum_{q \in \delta^-(p)} z_{qp} = z_{a_p}$. Consequently, Constraints (A.1a) and (A.1e) give lower and upper bounds on the flow of each arc in the network: the flow of arcs (q, p) with p being a non-depot node is bounded by $[\overline{v}_{jp}, 1]$; for the remaining arcs, the capacity of the ones whose tail is a non-depot node is 1; lastly, arcs between two depot nodes have infinite capacity. Furthermore, Constraints (A.1b) - (A.1d) correspond to flow conservation equations at every node of the network. Therefore, for variables z the above constraints a network flow

polyhedron with integer coefficients, and hence the optimal z is integral. □

Proof of Proposition 5. As Problem 5 is by definition the setting of Problem 1 with service radius management, it suffices to show that Constraint set (2.4) accurately models the service radius mechanics described in the formulation of Problem 5 and allows to compute the optimal service radii. Consider Algorithm 4, which partitions the set of orders

Algorithm 4 (R_PARTITION_SORT)

Input: $N, \{(r_j, \tau_j)\}_{j \in N}, \{t_\ell\}_{\ell=1}^{R}$
Output: Lists of orders $B_1, \ldots, B_R$, each sorted in ascending order of τ_j.

```
B_ℓ ← ∅, ∀ℓ = 1, ..., R
j ← 1
for ℓ ∈ {1, ..., R} do
    while r_j < t_{ℓ+1} do
        B_ℓ ← B_ℓ ∪ {j}
        j ← j + 1
    Sort elements of B_ℓ in ascending order of τ_j
return {B_ℓ}_{ℓ=1}^{R}
```

N into the R sorted lists $\{B_\ell\}_{\ell=1}^{R}$. Note that Constraint set (2.4) enforces that for each $\ell = 1, \ldots, R$, whenever order $B_{\ell,i}$ is served, so are all the orders $B_{\ell,j}, \forall j < i$, which is the definition of service radius. Moreover, this formulation allows us to compute the optimal service radius for each radius shift without using explicit decision variables for the service radii: let (v^*, z^*) be the optimal solution to Problem 5, and for each $\ell \in \{1, \ldots, R\}$, let $\mu_\ell = \max\{i : \sum_p v^*_{B_{\ell,i},p} = 1\}$, then by construction of B_ℓ each optimal radius is calculated as $\rho_\ell = \tau_{B_{\ell,\mu_\ell}}$. □

Proof of Proposition 6. Running Algorithm 4 for each depot $d \in \{1, 2\}$ and replacing τ_j by $\tilde{\tau}^d_j, \forall j \in N_d$ constructs the sorted lists $\{B^d_\ell\}_{\ell=1}^{R_d}$. Then the claim follows from applying the same argument for Proposition 5 to each depot. □

A.2 Construction of the time-expanded network for the *L*-star setting

For line segment $h \in \{1, 2, \ldots, L\}$, let $\mathcal{T}_0^h$ be the set of time points of possible dispatches to line segment h. Algorithm 5 constructs the time-expanded network for the L-star setting. This algorithm firstly creates depot nodes at ready times of every order in the system for all line segments, and then for each line segment h, it initializes the set $\mathcal{T}_0^h$ with the ready times of orders to be delivered along h (lines 1 - 5).

Subsequently in lines 6 and 7, Algorithm 5 creates arcs and non-depot nodes for dispatches at ready times, and arcs and depot nodes for the corresponding return times, by executing Algorithm 6 once for every line segment. For a given line segment h, this subroutine works similar to Algorithm 1 for the single line segment setting, although this extension also defines new dispatches from a return node to every line segments. This is done with the help of auxiliary sets $\mathcal{S}_0^{h'}$, which keep track of new dispatch times not yet in $\mathcal{T}_0^{h'}$.

Algorithm 5 performs a final iterative step in lines 8 - 14 if new dispatches are yet to be evaluated for some line segments, i.e. if $S_{temp} \neq \emptyset$. For a line segment $h \in S_{temp}$, Algorithm 6 is executed to evaluate and define new dispatch times in the set $\mathcal{S}_0^h$, and to define the corresponding nodes and arcs. Note that this in turn may generate new dispatch times for some other line segment h' due to new return times, in which case these are appended to $\mathcal{S}_0^{h'}$ and h' is included in S_{temp}. Once the new dispatches are defined, the new dispatch times are appended to the defined dispatch times $\mathcal{T}_0^h$, S_{temp} is computed again. This process repeats until no new dispatch times are left to be evaluated for any line segment, i.e. when $S_{temp} = \emptyset$. At this point, Algorithm 5 returns the network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$.

Algorithm 5 L_STAR_NETWORK_CREATION

Input: L, $\{N_h, \{(r_j, \tau_j, Q_j)\}_{j \in N_h}\}_{h \in \{1,2,\ldots,L\}}, T$
Output: Directed network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$

$\mathcal{V} \leftarrow \{(r_j, 0, 0)\}_{j \in N} \cup \{(T, 0, 0)\}$
$\mathcal{A} \leftarrow \emptyset$
for $h \in \{1, 2, \ldots, L\}$ **do**
    $\mathcal{T}_0^h \leftarrow \{r_j\}_{j \in N_h}$
    $\mathcal{S}_0^h \leftarrow \emptyset$
for $h \in \{1, 2, \ldots, L\}$ **do**
    $(\mathcal{V}, \mathcal{A}, \{\mathcal{S}_0^{h'}\}_{h'=1}^L) = \text{L\_STAR\_ROUTES}(\mathcal{V}, \mathcal{A}, h, \mathcal{T}_0^h, \{\mathcal{T}_0^{h'}\}_{h'=1}^L, \{\mathcal{S}_0^{h'}\}_{h'=1}^L)$
$S_{temp} \leftarrow \{h \in \{1, 2, \ldots, L\} : \mathcal{S}_0^h \neq \emptyset\}$
while $S_{temp} \neq \emptyset$ **do**
    Let h be one of the elements in S_{temp}
    $(\mathcal{V}, \mathcal{A}, \{\mathcal{S}_0^{h'}\}_{h'=1}^L) = \text{L\_STAR\_ROUTES}(\mathcal{V}, \mathcal{A}, h, \mathcal{S}_0^h, \{\mathcal{T}_0^{h'}\}_{h'=1}^L, \{\mathcal{S}_0^{h'}\}_{h'=1}^L)$
    $\mathcal{T}_0^h \leftarrow \mathcal{T}_0^h \cup \mathcal{S}_0^h$
    $\mathcal{S}_0^h \leftarrow \emptyset$
    $S_{temp} \leftarrow \{h' \in \{1, 2, \ldots, L\} : \mathcal{S}_0^{h'} \neq \emptyset\}$
return $\mathcal{N} = (\mathcal{V}, \mathcal{A})$

Algorithm 6 L_STAR_ROUTES$(\mathcal{V}, \mathcal{A}, h, \mathcal{T}, \{\mathcal{T}_0^{h'}\}_{h'=1}^L, \{\mathcal{S}_0^{h'}\}_{h'=1}^L)$

Input: Node set $\mathcal{V}$, arc set $\mathcal{A}$, line segment h, set of time point at the depot $\mathcal{T}$, sets of existing depot time points $\{\mathcal{T}_0^{h'}\}_{h'=1}^L$, sets of new returning time point at all line segments $\{\mathcal{S}_0^{h'}\}_{h'=1}^L$
Output: Updated sets $\mathcal{V}, \mathcal{A}, \{\mathcal{S}_0^{h'}\}_{h'=1}^L$

for $t \in \mathcal{T}$ **do**
    Find lowest $j^* \in N \cup \{n+1\}$ s.t. $t < r_{j^*}$ ▷ $r_{n+1} \equiv T$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t, 0, 0), (r_{j^*}, 0, 0))\}$
    Compute set of active orders $A_h(t) \subseteq N_h$
    Sort $\{\tau_j\}_{j \in A_h(t)}$ in ascending order, into $\{\tau_{(i)}\}_{i=1}^{|A_h(t)|}$
    for $i \in \{1, 2, \ldots, |A_h(t)|\}$ **do**
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + \tau_{(i)}, \tau_{(i)}, h)\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tau_{(i-1)}, \tau_{(i-1)}, \mathbf{1}_{[i \neq 1]} \times h), (t + \tau_{(i)}, \tau_{(i)}, h))\}$ ▷ $\tau_{(0)} \equiv 0$
        for $h' \in \{1, 2, \ldots, L\}$ **do**
            if $t + 2\tau_{(i)} \notin \mathcal{T}_0^{h'}$ **then**
                $\mathcal{S}_0^{h'} \leftarrow \mathcal{S}_0^{h'} \cup \{t + 2\tau_{(i)}\}$
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + 2\tau_{(i)}, 0, 0)\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tau_{(i)}, \tau_{(i)}, h), (t + 2\tau_{(i)}, 0, 0))\}$
return $(\mathcal{V}, \mathcal{A}, \{\mathcal{S}_0^{h'}\}_{h'=1}^L)$

A.3 Construction of the time-expanded network for the two-depot setting

The network construction procedure is presented in Algorithm 7. Here, the location of depot d in the x-axis is denoted as ξ_d, with $\xi_1 = 0$ and $\xi_2 = U$. In lines 1 - 4, the constructor first defines depot nodes with order ready times r_i^d at each depot d, and arcs $((0,0,1),(r_1^2,U,2))$ and $((T,U,2),(T,0,1))$ to ensure that the resulting network has a unique source node $(0,0,1)$ and a unique sink node $(T,0,1)$. Moreover, it also initializes the set $\mathcal{T}_{\xi_d}$ of depot times with ready times $\{r_i^d\}_{i=1}^{n_d}$.

Then in lines 5 - 8, Algorithm 7 runs Algorithm 8 once for each depot $d \in \{1,2\}$ to define the non-depot nodes where orders in N_d may be served, and additional depot nodes that correspond to courier returning times to d (lines 2 - 12 of Algorithm 8). Note that in this setting, each dispatch from d allows to travel beyond the furthest delivery location among the dispatched orders to get to the other depot $\overline{d}$, and so Algorithm 8 creates an extra arc $((t+\tilde{\tau}^d_{(|A_d(t)|)}, \tau^d_{(|A_d(t)|)}, d),(t+U,\xi_{\overline{d}},\overline{d}))$ from the furthest non-depot node of a dispatch to the corresponding arrival node at $\overline{d}$. The new arrival node to $\overline{d}$ is in turn a potential dispatch from that depot, and so the corresponding arrival time is stored in the set $\mathcal{T}^{potential}_{\xi_{\overline{d}}}$ so a dispatch can be evaluated later on (lines 13 - 15 of Algorithm 8).

Lastly, Algorithm 7 performs a final step in lines 9 - 14 that iteratively runs Algorithm 8 to evaluate new potential dispatch from each depot d at times in $\mathcal{T}^{potential}_{\xi_d}$ (similar to Algorithm 5 for the L-star setting). These dispatches may produce new arrival times to depot $\overline{d}$ due to crosses between depots, each of them in turn potentially defining a new dispatch time from $\overline{d}$. In such case, the new dispatch time is added to the set $\mathcal{T}^{potential}_{\xi_{\overline{d}}}$. Once the dispatches nodes and arcs are defined for a depot d, the new dispatch times are appended to the set of defined dispatch times $\mathcal{T}_{\xi_d}$. The iterative process repeats until no new dispatch times are discovered for either depot, namely, $\mathcal{T}^{potential}_{\xi_d} \subseteq \mathcal{T}_{\xi_d}$ for some $d \in \{1,2\}$.

Algorithm 7 (2_DEPOT_CREATE_NETWORK)

Input: $\{N_d, \boldsymbol{r}^d, \boldsymbol{\tau}^d, \boldsymbol{Q}^d\}_{d\in\{1,2\}}, T, U$
Output: Directed network $\mathcal{N} = (\mathcal{V}, \mathcal{A})$

$\mathcal{V} \leftarrow \bigcup_{d=1}^{2}\{(r_j^d, \xi_d, d)\}_{j=1}^{n_d} \cup \{(T, 0, 1), (T, U, 2)\}$
$\mathcal{A} \leftarrow \{((0, 0, 1), (r_1^2, U, 2)), ((T, U, 2), (T, 0, 1))\}$
$\mathcal{T}_0 \leftarrow \{r_j^1\}_{j=1}^{n_1}$
$\mathcal{T}_U \leftarrow \{r_j^2\}_{j=1}^{n_2}$
$(\mathcal{V}, \mathcal{A}, \mathcal{T}_0, \mathcal{T}_U^{potential}) \leftarrow$ 2_DEPOT_ROUTES$(\mathcal{V}, \mathcal{A}, 1, \mathcal{T}_0)$
$\mathcal{T}_U \leftarrow \mathcal{T}_U \cup \mathcal{T}_U^{potential}$
$\mathcal{T}_U^{potential} \leftarrow \emptyset$
$(\mathcal{V}, \mathcal{A}, \mathcal{T}_U, \mathcal{T}_0^{potential}) \leftarrow$ 2_DEPOT_ROUTES$(\mathcal{V}, \mathcal{A}, 2, \mathcal{T}_U)$
while True **do**
    for $d \in \{1, 2\}$ **do**
        if $\mathcal{T}_{\xi_d}^{potential} \subseteq \mathcal{T}_{\xi_d}$ **then return** $\mathcal{N} = (\mathcal{V}, \mathcal{A})$
        $(\mathcal{V}, \mathcal{A}, \mathcal{T}_{\xi_d}^{potential}, \mathcal{T}_{\xi_{\overline{d}}}^{potential}) \leftarrow$ 2_DEPOT_ROUTES$(\mathcal{V}, \mathcal{A}, 1, \mathcal{T}_{\xi_d}^{potential} \setminus \mathcal{T}_{\xi_d})$
        $\mathcal{T}_{\xi_d} \leftarrow \mathcal{T}_{\xi_d} \cup \mathcal{T}_{\xi_d}^{potential}$
        $\mathcal{T}_{\xi_d}^{potential} \leftarrow \emptyset$

Algorithm 8 2_DEPOT_ROUTES$(\mathcal{V}, \mathcal{A}, d, \mathcal{T}_{\xi_d}^{new_dispatches})$

Input: Node set $\mathcal{V}$, arc set $\mathcal{A}$, depot d, time point set $\mathcal{T}_{\xi_d}^{new\_dispatches}$
Output: Updated sets $\mathcal{V}$,$\mathcal{A}$,$\mathcal{T}_{\xi_d}^{new\_dispatches}$, and set of time points $\mathcal{T}_{\xi_{\overline{d}}}^{potential}$

$\mathcal{T}_{\xi_{\overline{d}}}^{potential} \leftarrow \emptyset$
for $t \in \mathcal{T}_{\xi_d}^{new\_dispatches}$ **do**
    Find lowest $j^* \in \{1, 2, \ldots, n_d + 1\}$ s.t. $t < r_{j^*}^d$ ▷ $r_{n_d+1}^d \equiv T$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t, \xi_d, d), (r_{j^*}^d, \xi_d, d))\}$
    Compute set of active orders $A_d(t) \subseteq N_d$
    Sort $\{\tau_j^d\}_{j\in A_d(t)}$ in ascending or descending order if $d = 1$ or $d = 2$, respectively (if $\{\tau_{(i)}^d\}_{i=1}^{|A_d(t)|}$ is the corresponding sorted sequence, then $\tau_{(i)}^1 \leq \tau_{(i+1)}^1$ and $\tau_{(i)}^2 \geq \tau_{(i+1)}^2, \forall i \in \{1, \ldots, |A_d(t)| - 1\}$)
    for $i \in \{1, 2, \ldots, |A_d(t)|\}$ **do**
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + \tilde{\tau}_{(i)}^d, \tau_{(i)}^d, d)\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tilde{\tau}_{(i-1)}^d, \tau_{(i-1)}^d, d), (t + \tilde{\tau}_{(i)}^d, \tau_{(i)}^d, d))\}$ ▷ $\tau_{(0)}^d \equiv \xi_d$
        $\mathcal{T}_{\xi_d}^{new\_dispatches} \leftarrow \mathcal{T}_{\xi_d}^{new\_dispatches} \cup \{t + 2\tilde{\tau}_{(i)}^d\}$
        $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + 2\tilde{\tau}_{(i)}^d, \xi_d, d)\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tilde{\tau}_{(i)}^d, \tau_{(i)}^d, d), (t + 2\tilde{\tau}_{(i)}^d, \xi_d, d)\}$
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(t + U, \xi_{\overline{d}}, \overline{d})\}$
    $\mathcal{T}_{\xi_{\overline{d}}}^{potential} \leftarrow \mathcal{T}_{\xi_{\overline{d}}}^{potential} \cup \{(t + U)\}$
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{((t + \tilde{\tau}_{(|A_d(t)|)}^d, \tau_{(|A_d(t)|)}^d, d), (t + U, \xi_{\overline{d}}, \overline{d}))\}$
return $(\mathcal{V}, \mathcal{A}, \mathcal{T}_{\xi_d}^{new\_dispatches}, \mathcal{T}_{\xi_{\overline{d}}}^{potential})$

Table A.1: Characterization of orders of numerical example

d	j	r_j^d	τ_j^d	$\tilde{\tau}_j^d$	Q_j^d
1	1	0	2	2	3
1	2	6	1	1	9
2	1	2	3	1	5

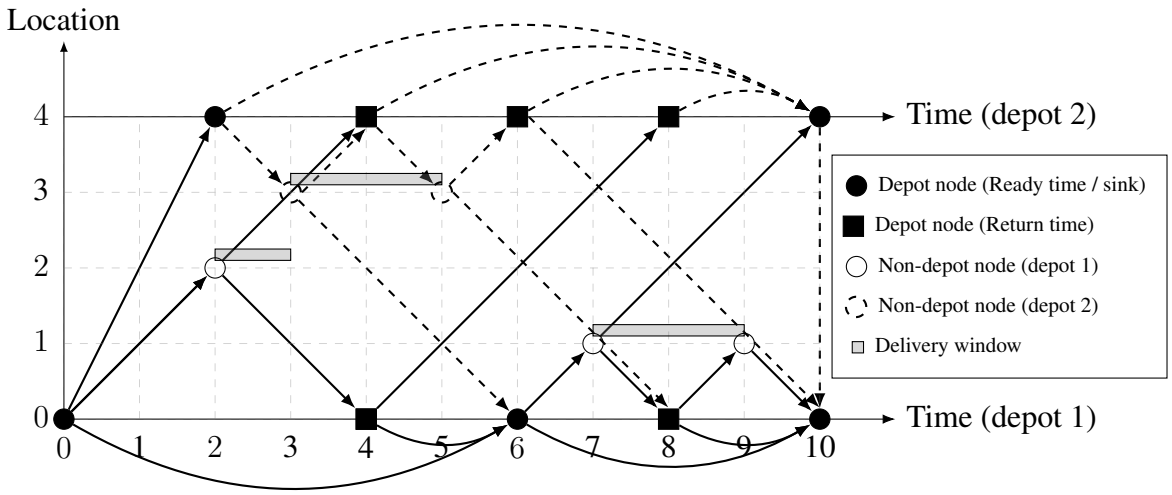


Figure A.1: Example of a time-expanded network with 2 depots and 3 orders. Solid arcs are associated to depot 1, and dashed arcs to depot 2

A numerical example. Figure A.1 illustrates a network example with two depots at locations 0 and $U = 4$, with parameter values of $S = 3$, $T = 10$, and orders information as in Table A.1. Depot nodes at location 0 and non-depot nodes with solid edge lines correspond to nodes associated to depot 1, whereas depot nodes at location 4 and non-depot nodes with dashed edge lines correspond to depot 2. The origin corresponds to node $(0, 0, 1)$. A flow of courier m is injected to the network through the origin node and arc $((0, 0, 1), (2, 4, 2))$ allows to selectively allocate couriers to begin their shift at either depot. From there couriers can traverse from one depot to the other when being dispatched to serve orders, and by the end they finish the shift at either $(10, 4, 2)$ or $(10, 0, 1)$; in any case, the insertion of arc $((10, 4, 2), (10, 0, 1))$ allows to consider a single sink. In this particular example, a single courier is sufficient to achieve full service.

APPENDIX B

ADDITIONAL MATERIAL FOR CHAPTER 3

B.1 Construction of base courier regions

This section briefly describes the construction procedure for base courier regions. The goal of this step is to design more compact courier operating areas so couriers may have specific knowledge of where they will spend their block, ultimately improving their satisfaction. At the core of this procedure, the sets of restaurants defining each of the courier regions are determined by solving a p-median optimization model [28]. A travel time-based dissimilarity is used as the objective to be minimized by the p-median model, to facilitate compactness of the resulting regions.

Mathematically, given a set of restaurants D, let $S(d, d')$ be pairwise dissimilarity between restaurants $d \in D$ and $d' \in D$, and let $p \geq 1$ be the number of regions to construct. This step determines a partition $\{D_i\}_{i=1}^{p}$ of the set of restaurants D and a centroid for each subset of restaurants D_i corresponding to one of its elements, such that the total sum of pairwise dissimilarities between the centroids and the restaurants assigned to each of them is minimized. To formulate the corresponding p-median problem, consider the following decision variables:

$$v_{d,d'} = \begin{cases} 1 & \text{if restaurant } d \in D \text{ is assigned to base courier region with centroid } d' \in D \\ 0 & \text{otherwise} \end{cases}$$

The construction of base courier regions is performed by solving the following integer

program:

$$\max \sum_{d \in D} \sum_{d' \in D} S(d, d') v_{d,d'} \tag{B.1a}$$

$$\text{s.t.} \sum_{d' \in D} v_{d,d'} = 1, \qquad \forall d \in D \tag{B.1b}$$

$$\sum_{d \in D} v_{d,d} = p \tag{B.1c}$$

$$v_{d',d} \leq v_{d,d}, \qquad \forall d \in D, \ \ \forall d' \in D \tag{B.1d}$$

$$v_{d,d'} \in \{0, 1\}, \qquad \forall d \in D, \ \ \forall d' \in D \tag{B.1e}$$

Objective (B.1a) minimizes the total sum of pairwise dissimilarities between the each of the p centroids and the elements allocated to the corresponding region. Constraint set (B.1b) requires that every restaurant is assigned to one of the p centroids. Constraint (B.1c) enforces the selection of p centroids from the set of restaurants (each defining a base different courier region). Lastly, Constraint set (B.1d) limits the assignment of restaurants to the selected centroids.

Given an optimal vector v^*, let $\{d_1, \ldots d_p\}$ be the set of selected region centroids, *i.e.*, be the set of restaurants d satisfying $v^*_{d,d} = 1$. Then for $i \in \{1, \ldots, p\}$, the base courier region r_i is initialized by setting $D_i = \{d \in D : v^*_{d,d_i} = 1\}$ as its set of restaurants.

To account for restaurant separation distance as well as order volume in the construction of base courier regions, in practice the clustering procedure is performed using $S(d, d') \doteq o_d \cdot \tau_{\ell_d, \ell_{d'}}$, $\forall d, d' \in D$, where o_d is the number of orders placed to restaurant $d \in D$ throughout $\boldsymbol{T}$.

REFERENCES

[1] Acosta, Inc. and Technomic, Inc., *The Why? Behind The Dine*™, https://www.acosta.com/news/convenience-is-key-for-us-diners-according-to-acosta-and-technomic-s-the-why-behind-the-dinetm, 2018.

[2] N. Agatz, *Demand management in e-fulfillment*, EPS-2009-163-LIS. 2009, hdl.handle.net/1765/15425.

[3] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Optimization for dynamic ride-sharing: A review," *European Journal of Operational Research*, vol. 223, no. 2, pp. 295–303, 2012, https://doi.org/10.1016/j.ejor.2012.05.028.

[4] C. Archetti and L. Bertazzi, "Recent challenges in routing and inventory routing: E-commerce and last-mile delivery," *Networks*, vol. 77, no. 2, pp. 255–268, 2021, https://doi.org/10.1002/net.21995.

[5] C. Archetti, D. Feillet, and M. G. Speranza, "Complexity of routing problems with release dates," *European Journal of Operational Research*, vol. 247, no. 3, pp. 797–803, 2015, https://doi.org/10.1016/j.ejor.2015.06.057.

[6] A. M. Arslan, N. Agatz, L. Kroon, and R. Zuidwijk, "Crowdsourced delivery—a dynamic pickup and delivery problem with ad hoc drivers," *Transportation Science*, vol. 53, no. 1, pp. 222–235, 2019, https://doi.org/10.1287/trsc.2017.0803.

[7] B. Asdecker and F. Zirkelbach, "What drives the drivers? a qualitative perspective on what motivates the crowd delivery workforce," *53rd Hawaii International Conference on System Sciences*, 2020, https://doi.org/10.24251/HICSS.2020.490.

[8] R. Auad, A. Erera, and M. Savelsbergh, "Using simple integer programs to assess capacity requirements and demand management strategies in meal delivery," *Optimization Online*, 2020, http://www.optimization-online.org/DB_FILE/2020/11/8125.pdf.

[9] R. Auad and P. Van Hentenryck, "Ridesharing and fleet sizing for on-demand multimodal transit systems," *arXiv preprint arXiv:2101.10981*, 2021, https://arxiv.org/abs/2101.10981.

[10] G. Berbeglia, J.-F. Cordeau, and G. Laporte, "Dynamic pickup and delivery problems," *European Journal of Operational Research*, vol. 202, no. 1, pp. 8–15, 2010, https://doi.org/10.1016/j.ejor.2009.04.024.

[11] A. A. Bertossi, P. Carraresi, and G. Gallo, "On some matching problems arising in vehicle scheduling models," *Networks*, vol. 17, no. 3, pp. 271–281, 1987, https://doi.org/10.1002/net.3230170303.

[12] C. M. Bishop, *Neural networks for pattern recognition*. Oxford University Press, 1995.

[13] N. Boland, M. Hewitt, L. Marshall, and M. Savelsbergh, "The continuous-time service network design problem," *Operations Research*, vol. 65, no. 5, pp. 1303–1321, 2017, https://doi.org/10.1287/opre.2017.1624.

[14] J. Brenton, S. Bowen, and S. Elliot, *Time to cook is a luxury many families don't have*, https://theconversation.com/time-to-cook-is-a-luxury-many-families-dont-have-117158, 2019.

[15] S. Bunte and N. Kliewer, "An overview on vehicle scheduling models," *Public Transport*, vol. 1, no. 4, pp. 299–317, 2009, https://doi.org/10.1007/s12469-010-0018-5.

[16] A. M. Campbell and M. W. Savelsbergh, "Decision support for consumer direct grocery initiatives," *Transportation Science*, vol. 39, no. 3, pp. 313–327, 2005, https://doi.org/10.1287/trsc.1040.0105.

[17] V. Capalbo, G. Ghiani, and E. Manni, "The role of optimization and machine learning in e-commerce logistics in 2030," *International Journal of Economics and Management Engineering*, vol. 15, no. 3, pp. 290–294, 2021, https://publications.waset.org/10011929/the-role-of-optimization-and-machine-learning-in-e-commerce-logistics-in-2030.

[18] D. Cattaruzza, N. Absi, and D. Feillet, "The multi-trip vehicle routing problem with time windows and release dates," *Transportation Science*, vol. 50, no. 2, pp. 676–693, 2016, https://doi.org/10.1287/trsc.2015.0608.

[19] X. Chen, M. W. Ulmer, and B. W. Thomas, "Deep q-learning for same-day delivery with vehicles and drones," *European Journal of Operational Research*, 2021, https://doi.org/10.1016/j.ejor.2021.06.021.

[20] X. M. Chen, H. Zheng, J. Ke, and H. Yang, "Dynamic optimization strategies for on-demand ride services platform: Surge pricing, commission rate, and incentives," *Transportation Research Part B: Methodological*, vol. 138, pp. 23–45, 2020, https://doi.org/10.1016/j.trb.2020.05.005.

[21] A. Cheng, *Millennials are ordering more food delivery, but are they killing the kitchen, too?* https://www.forbes.com/sites/andriacheng/2018/06/26/millennials-are-ordering-food-for-delivery-more-but-are-they-killing-the-kitchen-too/?sh=67b8e526393e, 2018.

[22] K. L. Croxton, D. M. Lambert, S. J. Garcia-Dastugue, and D. S. Rogers, "The demand management process," *The International Journal of Logistics Management*, vol. 13, no. 2, pp. 51–66, 2002, https://doi.org/10.1108/09574090210806423.

[23] I. Dayarian and M. Savelsbergh, "Crowdshipping and same-day delivery: Employing in-store customers to deliver online orders," *Production and Operations Management*, vol. 29, no. 9, pp. 2153–2174, 2020, https://doi.org/10.1111/poms.13219.

[24] A. Erera, M. Hewitt, M. Savelsbergh, and Y. Zhang, "Improved load plan design through integer programming based local search," *Transportation Science*, vol. 47, no. 3, pp. 412–427, 2013, https://doi.org/10.1287/trsc.1120.0441.

[25] H. Everett, *19 Online Ordering Statistics Every Restaurateur Should Know in 2019*, https://upserve.com/restaurant-insider/online-ordering-statistics/, 2019.

[26] P. C. Guedes and D. Borenstein, "Real-time multi-depot vehicle type rescheduling problem," *Transportation Research Part B: Methodological*, vol. 108, pp. 217–234, 2018, https://doi.org/10.1016/j.trb.2017.12.012.

[27] B. Guo, Y. Liu, L. Wang, V. O. Li, J. C. Lam, and Z. Yu, "Task allocation in spatial crowdsourcing: Current state and future directions," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1749–1764, 2018, https://doi.org/10.1109/JIOT.2018.2815982.

[28] S. L. Hakimi, "Optimum distribution of switching centers in a communication network and some related graph theoretic problems," *Operations research*, vol. 13, no. 3, pp. 462–475, 1965, https://doi.org/10.1287/opre.13.3.462.

[29] Y. Huang, L. Zhao, W. B. Powell, Y. Tong, and I. O. Ryzhov, "Optimal learning for urban delivery fleet allocation," *Transportation Science*, vol. 53, no. 3, pp. 623–641, 2019, https://doi.org/10.1287/trsc.2018.0861.

[30] D. Huisman, R. Freling, and A. P. Wagelmans, "A robust solution approach to the dynamic vehicle scheduling problem," *Transportation Science*, vol. 38, no. 4, pp. 447–458, 2004, https://doi.org/10.1287/trsc.1030.0069.

[31] W. C. Jordan and S. C. Graves, "Principles on the benefits of manufacturing process flexibility," *Management Science*, vol. 41, no. 4, pp. 577–594, 1995, https://doi.org/10.1287/mnsc.41.4.577.

[32] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014, https://arxiv.org/abs/1412.6980.

[33] M. A. Klapp, A. L. Erera, and A. Toriello, "The one-dimensional dynamic dispatch waves problem," *Transportation Science*, vol. 52, no. 2, pp. 402–415, 2016, https://doi.org/10.1287/trsc.2016.0682.

[34] ——, "The dynamic dispatch waves problem for same-day delivery," *European Journal of Operational Research*, vol. 271, no. 2, pp. 519–534, 2018, https://doi.org/10.1016/j.ejor.2018.05.032.

[35] R. Klein, S. Koch, C. Steinhardt, and A. K. Strauss, "A review of revenue management: Recent generalizations and advances in industry applications," *European Journal of Operational Research*, vol. 284, no. 2, pp. 397–412, 2020, https://doi.org/10.1016/j.ejor.2019.06.034.

[36] K. Lin, R. Zhao, Z. Xu, and J. Zhou, "Efficient large-scale fleet management via multi-agent deep reinforcement learning," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, https://doi.org/10.1145/3219819.3219993, 2018, pp. 1774–1783.

[37] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992, https://doi.org/10.1007/BF00992699.

[38] Y. Liu, "An optimization-driven dynamic vehicle routing algorithm for on-demand meal delivery using drones," *Computers & Operations Research*, vol. 111, pp. 1–20, 2019, https://doi.org/10.1016/j.cor.2019.05.024.

[39] Mintel, *9 in 10 us food delivery service users say it makes their lives easier*, https://www.mintel.com/press-centre/food-and-drink/9-in-10-us-food-delivery-service-users-say-it-makes-their-lives-easier, 2016.

[40] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013, https://arxiv.org/abs/1312.5602.

[41] A. Mor and M. Speranza, "Vehicle routing problems over time: A survey," *4OR*, vol. 18, no. 2, pp. 129–149, 2020, https://doi.org/10.1007/s10288-020-00433-2.

[42] Morgan Stanley Research, *Is online food delivery about to get "amazoned?"* https://www.morganstanley.com/ideas/online-food-delivery-market-expands/, 2017.

[43] Nation's Restaurant News, *Restaurant takeout and delivery are taking a bite out of dine-in traffic*, https://www.nrn.com/sponsored-content/restaurant-takeout-and-delivery-are-taking-bite-out-dine-traffic, 2016.

[44] V. Pillac, M. Gendreau, C. Guéret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1–11, 2013, https://doi.org/10.1016/j.ejor.2012.08.015.

[45] H. N. Psaraftis, M. Wen, and C. A. Kontovas, "Dynamic vehicle routing problems: Three decades and counting," *Networks*, vol. 67, no. 1, pp. 3–31, 2016, https://doi.org/10.1002/net.21628.

[46] Z. Qin, X. Tang, Y. Jiao, F. Zhang, Z. Xu, H. Zhu, and J. Ye, "Ride-hailing order dispatching at didi via reinforcement learning," *INFORMS Journal on Applied Analytics*, vol. 50, no. 5, pp. 272–286, 2020, https://doi.org/10.1287/inte.2020.1047.

[47] D. Reyes, A. Erera, M. Savelsbergh, S. Sahasrabudhe, and R. O'Neil, "The meal delivery routing problem," *Optimization Online*, vol. 6571, 2018, http://www.optimization-online.org/DB_FILE/2018/04/6571.pdf.

[48] D. Reyes, A. L. Erera, and M. W. Savelsbergh, "Complexity of routing problems with release dates and deadlines," *European Journal of Operational Research*, vol. 266, no. 1, pp. 29–34, 2018, https://doi.org/10.1016/j.ejor.2017.09.020.

[49] J. Saha, "An algorithm for bus scheduling problems," *Journal of the Operational Research Society*, vol. 21, no. 4, pp. 463–474, 1970, https://doi.org/10.2307/3008423.

[50] M. Savelsbergh and T. Van Woensel, "50th anniversary invited article—city logistics: Challenges and opportunities," *Transportation Science*, vol. 50, no. 2, pp. 579–590, 2016, https://doi.org/10.1287/trsc.2016.0675.

[51] S. Shead, *Covid has accelerated the adoption of online food delivery by 2 to 3 years, Deliveroo CEO says*, https://www.cnbc.com/2020/12/03/deliveroo-ceo-says-covid-has-accelerated-adoption-of-takeaway-apps.html, 2020.

[52] B. C. Shelbourne, M. Battarra, and C. N. Potts, "The vehicle routing problem with release and due dates," *INFORMS Journal on Computing*, vol. 29, no. 4, pp. 705–723, 2017, https://doi.org/10.1287/ijoc.2017.0756.

[53] D. Simchi-Levi and Y. Wei, "Understanding the performance of the long chain and sparse designs in process flexibility," *Operations research*, vol. 60, no. 5, pp. 1125–1141, 2012, https://doi.org/10.1287/opre.1120.1081.

[54] M. Skutella, "An introduction to network flows over time," in *Cook W., Lovász L., Vygen J. (eds) Research Trends in Combinatorial Optimization*, https://doi.org/10.1007/978-3-540-76796-1_21, Springer, 2009, pp. 451–482.

[55] M. Steingoltz and M. Picciola, *Meals on wheels: The digital ordering and delivery restaurant revolution*, https://www.lek.com/insights/ei/digital-restaurant-delivery, 2019.

[56] A. Strauss, N. Gülpınar, and Y. Zheng, "Dynamic pricing of flexible time slots for attended home delivery," *European Journal of Operational Research*, 2020, https://doi.org/10.1016/j.ejor.2020.03.007.

[57] A. K. Strauss, R. Klein, and C. Steinhardt, "A review of choice-based revenue management: Theory and methods," *European Journal of Operational Research*, vol. 271, no. 2, pp. 375–387, 2018, https://doi.org/10.1016/j.ejor.2018.01.011.

[58] R. S. Sutton, A. G. Barto, *et al.*, *Introduction to reinforcement learning*. MIT Press Cambridge, 1998, vol. 135.

[59] E. Uçar, Ş. İ. Birbil, and İ. Muter, "Managing disruptions in the multi-depot vehicle scheduling problem," *Transportation Research Part B: Methodological*, vol. 105, pp. 249–269, 2017, https://doi.org/10.1016/j.trb.2017.09.002.

[60] M. Ulmer, A. Erera, and M. Savelsbergh, "Dynamic service area sizing in urban delivery," https://www.researchgate.net/profile/Marlin-Ulmer/publication/341071310_Dynamic_Service_Area_Sizing_in_Urban_Delivery/links/5eabe37a92851cb267693434/Dynamic-Service-Area-Sizing-in-Urban-Delivery.pdf.

[61] M. Ulmer and M. Savelsbergh, "Workforce scheduling in the era of crowdsourced delivery," *Transportation Science*, vol. 54, no. 4, pp. 1113–1133, 2020, https://doi.org/10.1287/trsc.2020.0977.

[62] M. Ulmer, B. Thomas, and D. Mattfeld, "Preemptive depot returns for dynamic same-day delivery," *EURO Journal on Transportation and Logistics*, vol. 8, no. 4, pp. 327–361, 2019, https://doi.org/10.1007/s13676-018-0124-0.

[63] M. W. Ulmer, "Dynamic pricing and routing for same-day delivery," *Transportation Science*, vol. 54, no. 4, pp. 1016–1033, 2020, https://doi.org/10.1287/trsc.2019.0958.

[64] M. W. Ulmer and S. Streng, "Same-day delivery with pickup stations and autonomous vehicles," *Computers & Operations Research*, vol. 108, pp. 1–19, 2019, https://doi.org/10.1016/j.cor.2019.03.017.

[65] M. W. Ulmer, B. W. Thomas, A. M. Campbell, and N. Woyak, "The restaurant meal delivery problem: Dynamic pick-up and delivery with deadlines and random ready times," *Transportation Science*, 2020, https://doi.org/10.1287/trsc.2020.1000.

[66] ——, "The restaurant meal delivery problem: Dynamic pickup and delivery with deadlines and random ready times," *Transportation Science*, vol. 55, no. 1, pp. 75–100, 2021, https://doi.org/10.1287/trsc.2020.1000.

[67] Union Bank of Switzerland, *Is the kitchen dead?* https://www.ubs.com/global/en/investment-bank/in-focus/2018/dead-kitchen.html, 2018.

[68] S. A. Voccia, A. M. Campbell, and B. W. Thomas, "The same-day delivery problem for online purchases," *Transportation Science*, vol. 53, no. 1, pp. 167–184, 2019, https://doi.org/10.1287/trsc.2016.0732.

[69] D. M. Vu, M. Hewitt, N. Boland, and M. Savelsbergh, "Solving time dependent traveling salesman problems with time windows," *Optimization Online*, vol. 6640, 2018, http://www.optimization-online.org/DB_FILE/2018/05/6640.pdf.

[70] X. Wang, N. Agatz, and A. Erera, "Stable matching for dynamic ride-sharing systems," *Transportation Science*, vol. 52, no. 4, pp. 850–867, 2018, https://doi.org/10.1287/trsc.2017.0768.

[71] M. Woschank, E. Rauch, and H. Zsifkovits, "A review of further directions for artificial intelligence, machine learning, and deep learning in smart logistics," *Sustainability*, vol. 12, no. 9, p. 3760, 2020, https://doi.org/10.3390/su12093760.

[72] L. Yeo, *Which company is winning the restaurant food delivery war?* https://secondmeasure.com/datapoints/food-delivery-services-grubhub-uber-eats-doordash-postmates, 2021.

[73] B. Yildiz and M. Savelsbergh, "Provably high-quality solutions for the meal delivery routing problem," *Transportation Science*, vol. 53, no. 5, pp. 1372–1388, 2019, https://doi.org/10.1287/trsc.2018.0887.

[74] ——, "Service and capacity planning in crowd-sourced delivery," *Transportation Research Part C: Emerging Technologies*, vol. 100, pp. 177–199, 2019, https://doi.org/10.1016/j.trc.2019.01.021.

[75] ——, "Pricing for delivery time flexibility," *Transportation Research Part B: Methodological*, vol. 133, pp. 230–256, 2020, https://doi.org/10.1016/j.trb.2020.01.004.

[76] O. Zaleski, *Restaurants shrink as food delivery apps get more popular*, https://www.bloomberg.com/news/articles/2018-10-29/restaurants-shrink-as-food-delivery-apps-get-more-popular, 2018.

[77] H. Zhong, R. W. Hall, and M. Dessouky, "Territory planning and vehicle dispatching with driver learning," *Transportation Science*, vol. 41, no. 1, pp. 74–89, 2007, https://doi.org/10.1287/trsc.1060.0167.